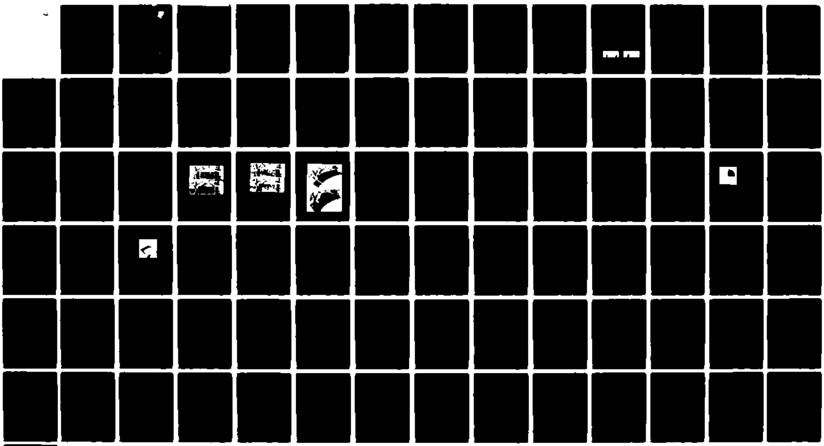
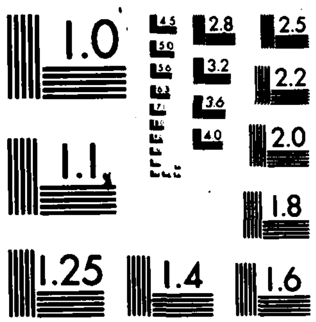
AD-A138 890 PRELIMINARY DSRS DESIGN(U) STANFORD UNIV CA DEPT OF COMPUTER SCIENCE T O BINFORD ET AL. SEP 83 RADC-TR-83-216 F30602-81-C-0169 1/1

UNCLASSIFIED F/G 8/2 NL



END
DATE
FILMED
DTIC




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A138890

RADC-TR-83-216
Final Technical Report
September 1983

# PRELIMINARY DSRS DESIGN


Stanford University

Thomas O. Binford and H. Harlyn Baker


APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED



ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

84 03 12 010

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-83-216 has been reviewed and is approved for publication.

APPROVED:

JOHN T. BOLAND
Project Engineer

APPROVED:

THADEUS J. DOMURAT
Acting Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER:

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC ( IRRA ) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>RADC-TR-83-216 | 2. GOVT ACCESSION NO.<br>AD-A138896 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>PRELIMINARY DSRS DESIGN | | 5. TYPE OF REPORT & PERIOD COVERED<br>Final Technical Report<br>1 Jun 82 - 1 Apr 83 |
| | | 6. PERFORMING ORG. REPORT NUMBER<br>N/A |
| 7. AUTHOR(s)<br>Thomas O. Binford<br>H. Harlyn Baker | | 8. CONTRACT OR GRANT NUMBER(s)<br>F30602-81-C-0169 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Department of Computer Science<br>Stanford University<br>Stanford CA 94305 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>67743-15<br>R25600P3 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Rome Air Development Center (IRRA)<br>Griffiss AFB NY 13441 | | 12. REPORT DATE<br>September 1983 |
| | | 13. NUMBER OF PAGES<br>86 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)<br>Same | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE<br>N/A |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: John T. Boland (IRRA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Correlation
Stereo Mapping

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes Stanford's approach to stereo mapping. This includes geometric modeling, computer vision and rule synthesis (with mono and stereo information). Also described are an improved edge operator (by Mairmont) and Stanford's analysis of epipolar geometry. A source code listing (in SAIL) of the epipolar analysis program is included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Contributions to this research have been made by Harlyn Baker, Thomas Binford, Cary Gray, Jitendra Malik, David Marimont, and Jean-Frederic Meller.

# TABLE OF CONTENTS

# 1: Demonstration of Stereo Mapping Technology

## *1.1 Introduction*

The focus of the research over this postdoctoral contract has been in augmenting and refining the specification of image matching strategies to be employed in an advanced stereo mapping system. Initially, our research task was to evaluate a preliminary design for a mapping system, and then by carrying out test implementations of elements of this proposed system, to iterate on the design process. As part of a continuing mapping program, this phase of the research would lead to more effective representations and improved matching strategies for surface reconstruction. The test implementations would build to provide a foundation on which we would develop an experimental rule-based stereo matching system. As part of this experimental system, we undertook to develop an interactive test facility to provide hand synthesized data for rule analysis. A change in the scope of the project occurred shortly after we had begun the research, and necessitated a redirection in the emphasis; rather than evaluating or continuing the design, we were to:

a) experiment with an existing matching system (showing its capabilities and assessing its applicability to the planned system);

b) demonstrate the design and the utility of the rule-based approach to surface inference from monocular information;

c) develop tools to support an interactive test facility.

## *1.2 Scope of the Research*

In a), we undertook to apply an existing stereo mapping system [Baker 1981] to some new imagery. This was designed to demonstrate its effectiveness, to expose its limitations, and to suggest both its role in an advanced mapping system and complementary research needed to improve its utility. Significant restructuring of the system was called for in enabling it to process this imagery. Details of these changes are described in section 5, which deals with the matching process. The modifications have now been implemented, enabling the system to:

- function on the output of an improved edge operator [Marimont 1982];
- use edge extent as one of its parameters in seeking optimized correspondence;
- exploit prepared transform information in processing images whose epipolar lines are not collinear with the scanning axes of the cameras.

Separate funding sources supported research in analysis and synthesis of rules for inference of three-dimensional shape from single images. This inference also has application in constraining search for matches in stereo correspondence. We will describe here some results of this research, and, as part of the demonstration, will show its use in shape inference and search constraint in the domain of Orthogonal Trihedral Vertices (OTV's).

Our work in developing tools has centered on:

a) a system for the hand construction of edge descriptions from hard copy imagery;

b) an interactive system for determining the transform to bring image pairs into collinear epipolar registration.

Both of these systems make extensive use of interactive graphics, and the latter takes much advantage of previous stereo research from our laboratory ([Gennery 1980]).

These are the three principal areas to be covered in this report:

- experimentation with the automated mapping system;
- assessment of rule synthesis from manual and automated edge processes;
- development of test facility tools for edge extraction and image registration.

## *1.3 Specific Tasks*

We will describe results in these areas of the research through discussion of the following:

a) implementation of the digitizing test facility;

b) the use of image edge descriptions produced using this facility and from an automated process [Marimont 1982] in synthesizing rules for stereo matching;

c) examination of Orthogonal Trihedral Vertex (OTV) inference rules;

d) development of a system for registration of image pairs;

e) analysis of stereo imagery with an automated stereo process.

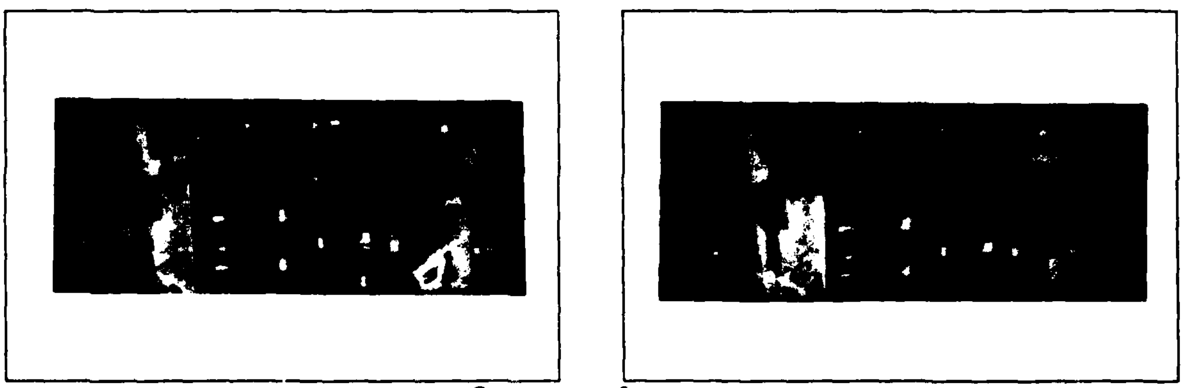
# 2: Digitizing Facility

## *2.1 Background*

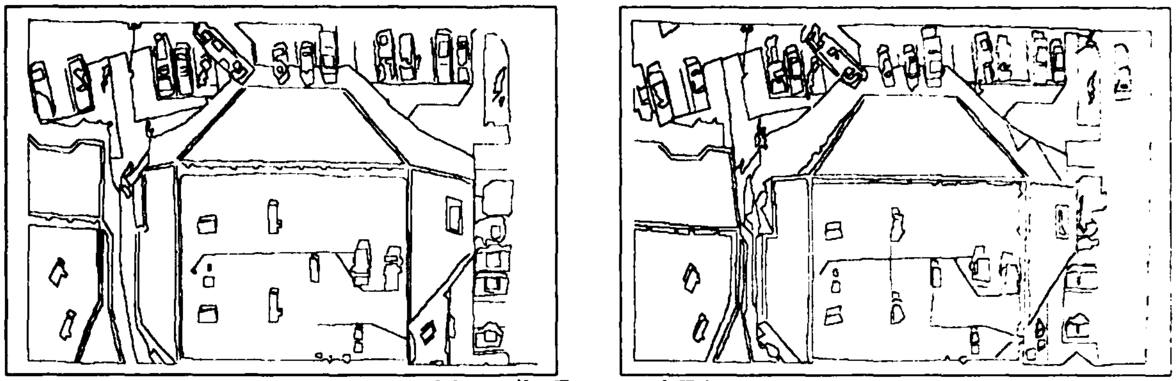
Our approach to rule development begins with hand synthesized and some automatically generated edge data. We have systems for the automatic generation of edge data (*i.e.* [Marimont 1982]). This data, extracted from a sufficently wide selection of imagery types, gives good insight into the current capabilities of automated processes. Automated processes, however, are not able presently to give as meaningful a description of an image as we would like, and have not been designed to provide the aggregrated abstractions research systems ([Lowe 1982]) will be soon supplying. To bridge this inadequacy, we work with both automatically generated data (the current state-of-the-art), and hand generated data (representative of the next generation of edge analysis processes). The hand generated data is obtained from a manually operated digitizing tablet. We have written a graphics-based digitizing and editing system to run with a GTCO tablet in producing these image descriptions. Section 6 lists instructions for the use of this program with the bitpad (GTCO tablet), and section 9 contains a source listing of the program.

## *2.2 Imagery Analysis with the Tablet*

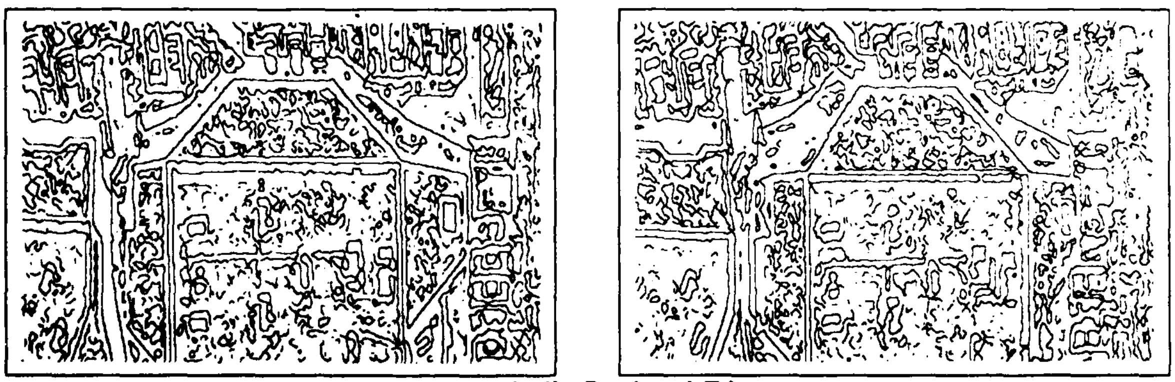
Figure 2-1 below shows an image pair of a building complex (referred to as the Sacramento imagery). Figure 2-2 shows the results of bitpad edge extraction on these images. Figure 2-3 shows the results of the Marimont operator [Marimont 1982] *on the image pair.* Manually generated edge data was produced using this facility for the analysis of rule synthesis of section 3. It was also used to digitize the building data of figures 2-1 for input to the OTV inference process, as figure 3-1.



*Sacramento Imagery*
Figure 2-1



*Manually Extracted Edges*
Figure 2-2



*Automatically Produced Edges*
Figure 2-3

# 3: Inference and Modelling

## *3.1 Introduction*

We have been addressing inference of matching rules and the use of model-based analysis both with theoretical analyses and with hand and automated analyses of specific matching strategies; the latter applied to both real and synthesized imagery examples.

We have obtained extended edge data from hand and automated processing for use in synthesis of matching rules. Results from earlier work on OTV analysis (orthogonal tr dr l vertices) have been exploited, and a demonstration of the use of such a rule for shape rence and in constraining search for correspondence has been prepared. Other funding has su rted research in representation of generic structures. We have taken examples from this moc research to produce ground and aerial views of a building complex, and have used this, as w other data, in rule synthesis.

## *3.2 Geometric Modelling and Computer Vision*

### 3.2.1 – Modelling, prediction and interpretation

Of course, one of the primary goals of research in computer vision is the development of systems that can recognize and locate objects in images. In order to identify such an object, it is clearly necessary to have some description of its characteristics that can be detected in an image. A ***representation*** of an object is the form this description takes.

One approach to representation is to provide the system with three-dimensional models of objects. Rotation of these models will allow objects to be observed, conceptually, from differing viewpoints. If parameters in a particular model are allowed to vary it is possible to have that single model represent a whole class of objects; constraining the parameters functions to delimit sub-classes. Further model manipulations, such as partitioning and projection, can be used to aid in mapping model to imagery data. The information contained in such object models may be used to determine possible interpretations of image features (*e.g.* , edges, ribbons, corners) and to provide feedback to predict the locations of such features in an image.

ACRONYM [Brooks 1981] is a three-dimensional rule-based modelling/vision system developed here at Stanford that provides, among other things, such feature prediction, model manipulation, and image interpretation. The rule-base operates on the models and on the sensed data to accomplish scene interpretation. Such a rule-based approach has been shown to be an effective form for constraint and search implementation, and allows easy modification and addition of new rules without the need of altering the underlying code.

Our group's intention over the next few years is to build a rule-based stereo system operating within ACRONYM whose functioning will include model-based prediction. Working toward this, we have been carrying out experiments on scene inference and model-based prediction that will lead to a repertoire of stereo matching rules.

### 3.2.2 – Models and stereo matching

One of the major difficulties in determining stereo correspondence is in dealing with the large number of matches that are possible. Solution is generally found by search through a large parameter space, where possible correspondences are limited by geometric or photometric constraints. Search can be reduced even more dramatically by endowing the matcher with broad domain specific and domain independent knowledge. Such knowledge can be rule-based and model-based. Our proposition here is that the three-dimensional information in object models, along with inference and prediction mechanisms, can be used to interpret features in image pairs. These interpretations can then be used as filters to constrain the matching. We demonstrate this notion with the example of Orthogonal Trihedral Vertices, often referred to as cube corners or OTVs. Other rules synthesized from analysis of both manually extracted and automated edge processes follow.

The work on cube corners points to additional usefulness for a model-based approach. OTV orientation analysis (from matches across pairs of views) yields almost complete solution for camera parmameters; constraints on sizes (again, from rules and models) could complete the camera solution. But the orientation information yielded by a match of a pair of vertices is valid only if the vertex is a cube corner; thus it is necessary to be able to distinguish between vertices that are cube corners and those that are not. If the models contain sufficient information to identify cube corners, then the problem of determining cube corners independently of the identification process is eliminated. In fact, both the search for cube corners and the search for identification are likely to be reduced when they are combined.

### 3.2.3 – OTV rule-based analysis

#### *OTV theory*

In cultural scenes, we find a large number of interior and exterior corners of cubes -- typically when two walls at right angles meet the roof or the floor. The importance of utilizing this common structural element the Orthogonal Trihedral Vertex (OTV) has been emphasized earlier[Liebes 1981] .Since they provide a very tight constraint -- the three edges are mutually orthogonal in space -- it is possible to calculate the three dimensional orientations given the projections in the image. This can be done for both orthographic and perspective viewing.

If the eye is assumed to be focussed on the vertex of the cube corner, perspective can be ignored and the projection of a cube corner in $XYZ$ space will simply be its orthogonal projection on the $XY$ plane. Suppose that some 3 - star has angles between its rays $a$, $b$ and $c$ and also that the rays are represented by the unit vectors $v_1$, $v_2$, $v_3$. We are interested in detecting whether there are three vectors in $XYZ$ space, which are mutually orthogonal and project, respectively to $v_1$, $v_2$, $v_3$.

Since projection is accomplished by dropping the $z$-component, any 3 such vectors must be of the form $v_1 + \lambda_1 z$, $v_2 + \lambda_2 z$, and $v_3 + \lambda_3 z$ where $z$ is the unit vector in the $z$-direction.

Requiring mutual orthogonality implies that the dot products of these vectors in pairs be zero. From these conditions and some simple manipulations we can calculate the formulas for

$$\lambda_1 = \pm\sqrt{-\frac{(\cos a)(\cos c)}{(\cos b)}}, \qquad \lambda_2 = \pm\sqrt{-\frac{(\cos a)(\cos b)}{(\cos c)}}, \qquad \lambda_3 = \pm\sqrt{-\frac{(\cos c)(\cos b)}{(\cos a)}}$$

Hence solutions exist if

a) $\cos a$, $\cos b$, $\cos c$ are all non-zero and

b) either one or three of $\cos a$, $\cos b$, $\cos c$ are negative, so that the quantities under the square root sign are positive.

These results were first derived in [Perkins 1968].

Thus we have a way of both eliminating false candidates for being OTVs and finding the 3-D orientations of valid OTV's. This algorithm has been implemented and run on data from the digitizing tablet.

***OTV with/from models***

Our analysis begins on both images, processing bottom up on the two images separately. As the rule system identifies likely OTVs in images (from its models), it proceeds to match them. The system should already have a tentative identification of the buildings containing the OTVs, so there should be relatively few possible matches at this point. Only OTVs that could be the same point on the same object need be compared. The analysis results in depths of matched objects, for all those objects having OTVs.

This requires that the modelling system handle point elements, and that it include both:

- inferring OTVs from models (volumes);
- accessing OTVs stored explicitly with the models.

## *3.3 Rule Synthesis*

### 3.3.1 – Inference rules

We continue with the development of inference rules. This work is a logical extension of previous work [Binford 1981, Lowe 1982] done at Stanford in developing rules for inferring surface information from a single view. General assumptions about illumination, object geometry, the imaging process etc. have been used to derive rules for making specific inferences. For stereo vision Arnold and Binford [Arnold 1980] have developed conditions on correspondence of edge and surface intervals. We divide our rules into two categories: monocular rules, which enable surface inference from a single view; and stereo rules, which facilitate cross-image matching.

### 3.3.2 – Monocular and stereo rules.

1. Monocular rules – Rules which have been developed for inferences from monocular views can be utilised to provide a partial 3-dimensional interpretation which directs search in the second view. This category includes the rule for interpretation of Orthogonal Trihedral Vertices.

   Another example is the T-junction rule [Binford 1981] which states that *'In absence of evidence to the contrary, the stem of a T is not nearer than the top, i.e. is coincident in space or further away'*. Application of this rule gives a set of nearer/farther relations. A hypothesized correspondence of edges which leads to inconsistent conclusions from the two views can be pruned from the search.

   An image line which is straight must be the image of a straight space curve unless the curve is planar and the observer is coincidentally aligned with the plane of curvature. This enables us to dismiss correspondences between straight edges in one view and curves in the other view. If two image curves are projectively consistent with parallel, we assume they are images of curves which are parallel in space. That implies that their images in the other view would be parallel *i.e.* parallels map to parallels.

   As these examples illustrate, most of the rules in [Binford 1981, Lowe 1982] and others developed by Malik and Binford have as direct corollaries stereo rules for checking the legality of a match. They can even direct the search process.

2. Stereo Rules – these are rules which have been derived from the stereo imaging process, and are a function of the imaging geometry.

   An example rule in this class, which has long been used for finding stereo correspondences, is the epipolars rule – corresponding points must lie on corresponding epipolar lines. These rules have inherently no monocular analogs. Here are a few:

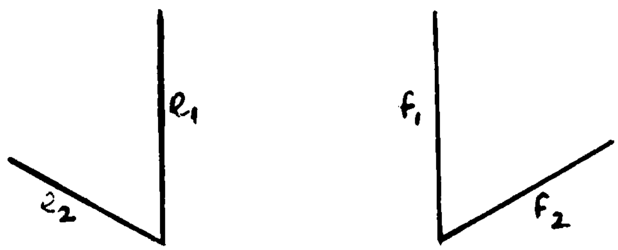
   a) Horizontal planes in one view get mapped to horizontal planes in the other view.

   b) Use of projective and quasi-projective invariants. This has not been examined in detail. Duda and Hart[Duda 1973] devote a chapter to this topic which has not really been exploited in stereo work.

   c) Conditions on correspondence of edges and surface intervals[Arnold 1980].

   d) Surface Occlusion rules:

      Surfaces visible in one view can be occluded in the other view. We are interested in the conditions when this takes place. The basic idea is that if we cross a surface, an obscuration of edge occurs. A left surface visible in a right view is visible in the left view unless there is obscuration by a tall object. Similarly a right surface visible in a left view will be seen unless obscured by a tall object. These surface-obscuration rules can be formalized by the cross-product rule:

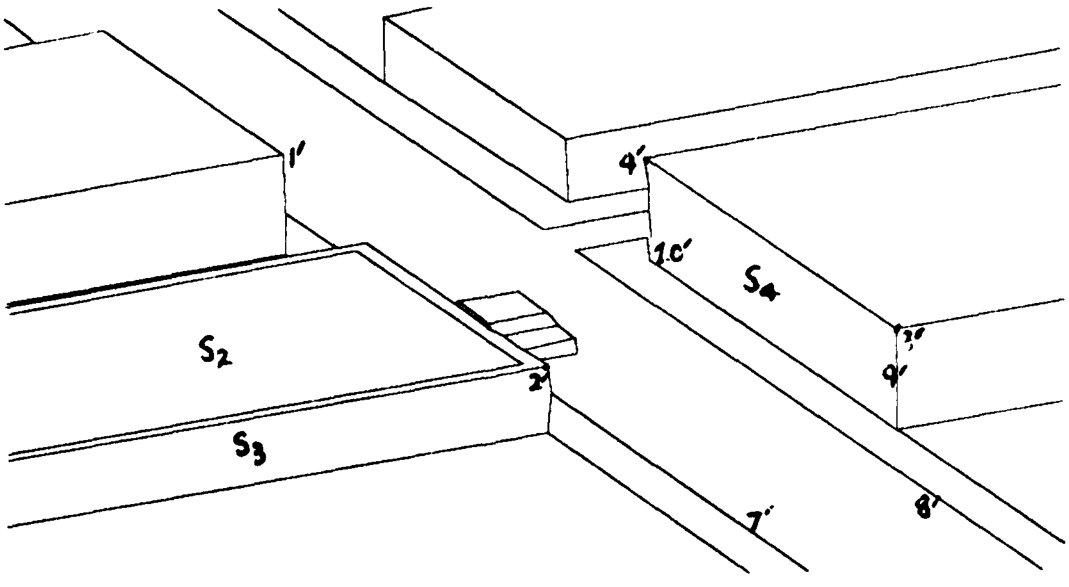



      For the hypothesized edge match $e_1$ with $f_1$ and $e_2$ with $f_2$, we compute the Z-component of the vector cross-product in the left image pair and the right image pair. If the z-components have opposite signs, we are seeing opposite sides of the surface. That implies that the object is not opaque.

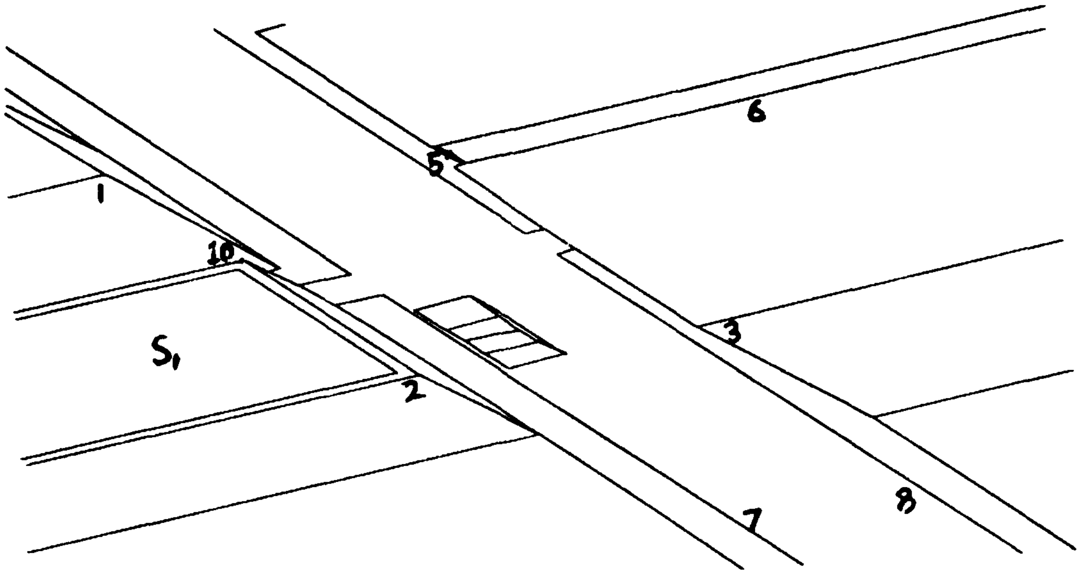
**3.3.3 – Use of inference rules in a test analysis**

Our preliminary results indicate good potential for the success of this approach. On hand simulations with line drawings of stereo pairs, the rules helped narrow down the choices considerably.

Consider the imagery shown in figures 3–1 and 3–2 Figure 3–1 is the right view and 3–2 the left view. Vertices 1, 2, 3 are orthogonal trihedral vertices. Using the formulae developed earlier, we can find the 3-D orientations of the edge vectors. These can be matched with the 3-D orientations of 1′, 2′, 3′ to obtain a registering of these vertices when combined with the epipolar constraint. All OTV's in one view need not be visible in the other *i.e.* 4′. Of the monocular constraints, the other major constraints which can be seen here are the T-junction rule and the parallels rule. In figure 3-2 edge 5 is behind edge 6. Edges 7 and 8 are parallel and so are 7′ and 8′. A match of 8 with 9′ would not be accepted. Surfaces $S_1$ and $S_2$ are both horizontal planes (as can be deduced from the OTV analysis) and can be matched. Surface $S_3$ is not horizontal. Here of course, this does not provide any new information. Surface $S_4$ being a left face in a left view is not guaranteed to be visible in the other view - as in fact it is. The cross-product rule could be used to dismiss a match between 10 and 10′.




Description of Left Image of Stereo Pair
Figure 3-1




*Description of Right Image of Stereo Pair*
**Figure 3-2**

# 4: Image Registration

## *4.1 Introduction to Epipolar Geometry*

The search process in automated stereo mapping can be greatly restricted, and computation times significantly reduced, if information is available relating the relative camera geometries of a stereo pair. Often this information is available in the reconnaisance data (or at least a rough approximation to it). Other manual and automated schemes have been devised to provide the information when it is not present with the imagery ([Gennery 1980], [Hallert 1960]). This camera geometry information allows establishing epipolar correspondence of lines across images. When this has been done, search in one image for match points of a feature in the other image can be constrained along a single vector. More generally, any features lying along a particular vector in the one image may be found along a single vector in the other image. These image plane vectors are termed epipolar lines. Corresponding vectors are termed corresponding or conjugate epipolar lines.

In this section we detail an algorithm for determining conjugate epipolar lines in a set of imagery for which such camera geometry information is not explicitly available. Here, we rely upon an operator to select corresponding points in the two images. The system automatically improves the resolution of the correspondence through Fourier interpolation over a match window [Gennery 1980]. The set of such points is taken by an automated camera solver to produce the needed geometric information. This point selection is done with pan/zoom cursor control on a graphics device. If the camera information is available (as, for example, from reconnaisance data), then the point matching phase may be omitted (although this provision has not been enabled in the current system). Equally, rough camera geometry information, if available, may be used to partially automate the point selection phase, although again this is not implemented here. [Gennery 1980] and [Moravec 1980] have implemented totally automatic camera solvers in their stereo matching systems. Our next improvement to this system will be to incorporate the image sampling and feature matching of the [Gennery 1980] system, removing the need for manual point selection.

## 4.2 Glossary of Terms

Given two cameras $C_1$ and $C_2$ with origins $\theta_1$ and $\theta_2$ and focal planes $P_1$ and $P_2$, we call:

Bundle of lines: A family of lines containing a common point.

Epipolar coordinates of a point: The number of the epipolar line it belongs to, and its distance to a fixed reference, like the epipole if it exists.

Epipolar direction: The direction of all epipolar lines if they are parallel.

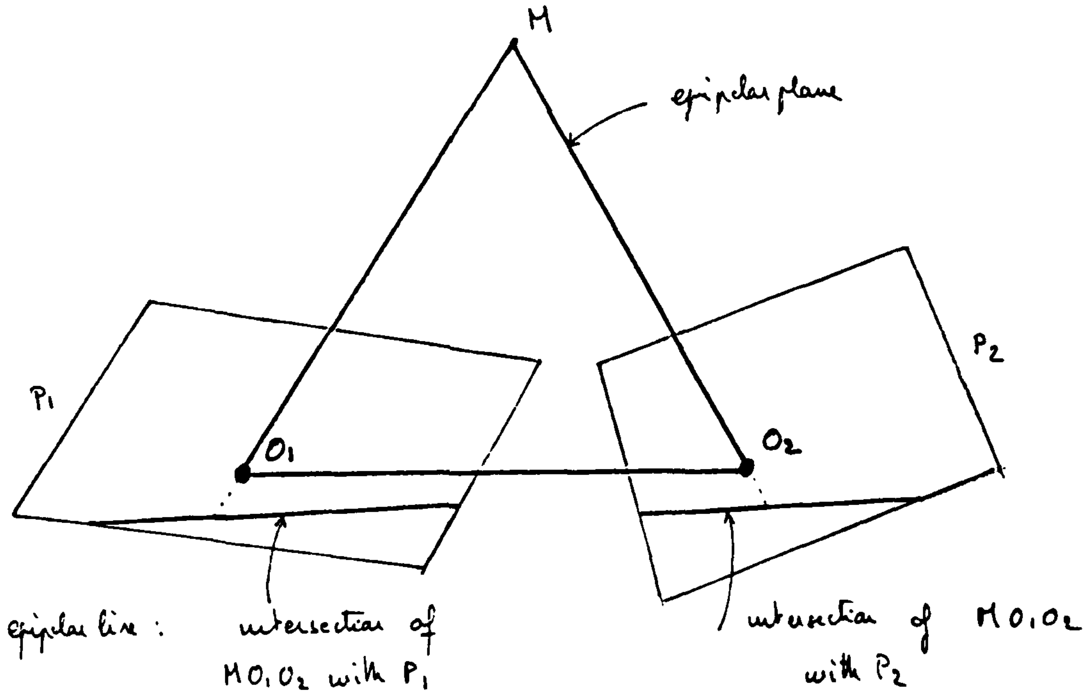
Epipolar line: The intersection of an epipolar plane with a focal plane. Alternate definition: the image in one camera of the pre-image of a point in the other camera's focal plane.

Epipolar plane: Any plane containing the two camera centers $\theta_1$ and $\theta_2$.

Epipolar space: A space where the coordinates are the epipolar coordinates. In this space, a horizontal line is an epipolar line, and the epipole, if it exists, is a whole vertical line.

Epipole: The intersection, if it exists, of all epipolar lines in a focal plane.

Conjugate epipolar lines: the intersections of an epipolar plane with the two focal planes.




Epipolar Geometry
Figure 4-1

## *4.3 Background Theory*

Given two stereo images $P_1$ and $P_2$ (the content of the focal planes $P_1$ and $P_2$ of the cameras), and two lines $L_1$ and $L_2$ contained in $P_1$ and $P_2$, in general every point of $L_1$ maps to a line segment in $P_2$, and there is no particular relationship between the line segments mapping to different points.

Fundamental property:

Every point in $L_1$ will map to a line segment contained in the same line $L_2$ if and only if $L_1$ and $L_2$ are a pair of conjugate epipolar lines.

Epipolar Geometry:

The family of epipolar lines in a focal plane is:

Either

a) a set of parallel lines having a common epipolar direction,

or

b) a bundle of lines, the intersection of which is the epipole.

## *4.4 Requirements of the System*

Given a pair of stereo images, we want to:

1) identify the kind of epipolar geometry present in the images;

2) explicitly show the epipolar lines belonging to each image;

3) for each image, compute the parameters which relate the original coordinates to the epipolar coordinates;

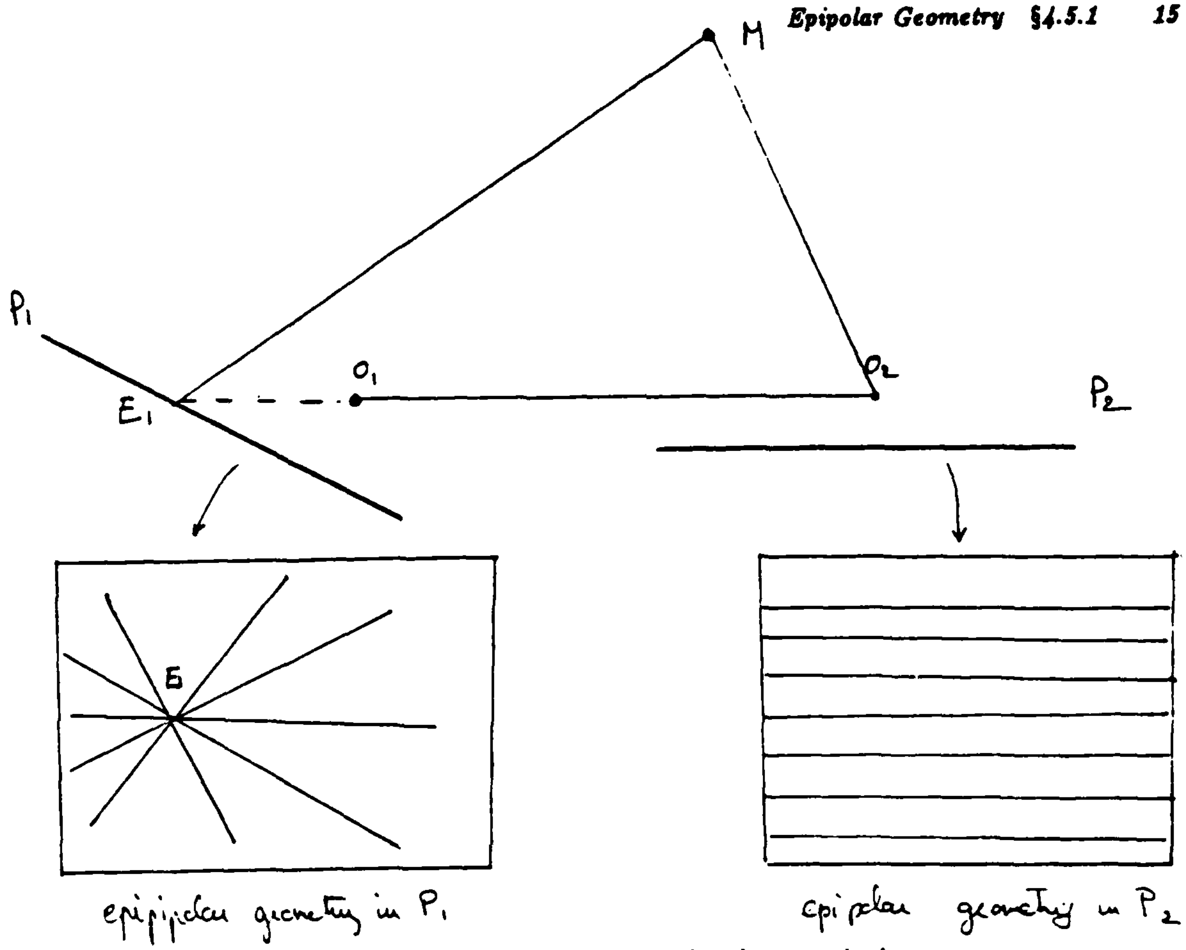
4) construct the image transforms in epipolar space.

Prior to these 4 steps, we will need to solve for the cameras, that is, to determine the 5 parameters describing their relative orientation. A procedure developed at Stanford [Gennery 1980] is used for this. We proceed as above, using simple analytic geometry for our calculations.

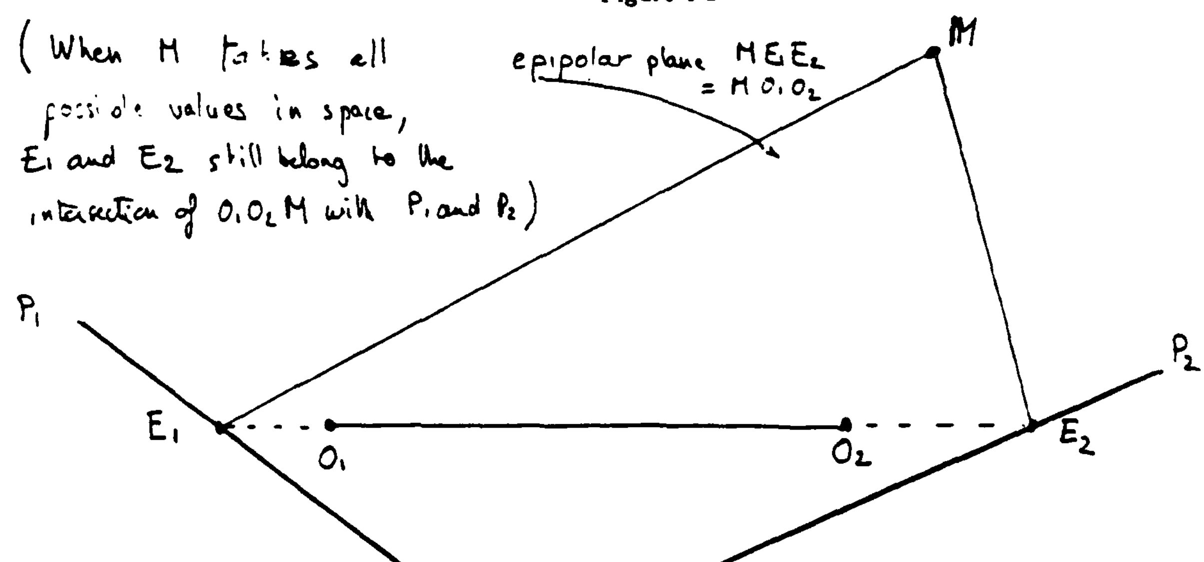
## *4.5 Algorithm Used*

### 4.5.1 – Camera registration output

Each camera is viewed as a referential $(\theta_i, x, y, z)$, $i \in \{1, 2\}$. The registration procedure yields *azimuth, elevation, pan, tilt,* and *roll* of one camera with respect to the other. From there, we compute:




*Top view of a situation with* only one *epipole*
**Figure 4-2**




*Top view of a situation with epipoles*
**Figure 4-3**

1) the rotation matrix $R$ between the two referentials:

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

2) the translation unit vector t, the components of which are:

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \text{in the base } \theta_1 xyz, \quad \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} \text{in the base } \theta_2 xyz$$

Note that the magnitude of the translation vector cannot be determined from a pair of images.

**4.5.2 – Epipolar geometry determination**

The focal planes $P_i$ are planes parallel to $\theta_i xy$, intersecting $\theta_i z$ at $z = f_i$, the focal distance. There is an epipole in plane $P_i$ if and only if the translation vector intersects this plane, that is, if and only if its third component is not zero.

We thus determine the case we are working with:

| | |
|---|---|
| if $\gamma \neq 0$ and $\nu \neq 0$, there are two epipoles: | CASE 1 |
| if $\gamma \neq 0$ and $\nu = 0$, there is one epipole $E_1$: | CASE 2 |
| if $\gamma = 0$ and $\nu \neq 0$, there is one epipole $E_2$: | CASE 3 |
| if $\gamma = 0$ and $\nu = 0$, there are no epipoles: | CASE 4 |

**Remarks:**

a) $\gamma = 0$ is replaced in the code by $|\gamma| <$ threshold, where threshold is chosen as a function of the arithmetic precision of the machine: if we had infinite precision, then we could consider every case as being case 1. Here threshold $= .0001$ was found to be a good estimate.

b) Most image pairs will belong to the first case, with $\gamma$ and $\nu$ of the order of .1. The epipoles exist, are outside the picture frame, and, for the images worked with to date, tend to be at a distance of about 10 times the picture dimension.

### 4.5.3 – Epipoles and epipolar directions

If the epipole $E_i$ exists, it is the extremity of the vector collinear to the translation vector, with a third component equal to $f_i$. If it does not, then the translation vector is the epipolar direction. Hence:

$$
\begin{array}{lcc}
\text{Case 1:} & E_1(\frac{\alpha f_1}{\gamma}, \frac{\beta f_1}{\gamma}) & E_2(\frac{\lambda f_1}{\nu}, \frac{\mu f_1}{\nu}) \\
\text{Case 2:} & E_1(\frac{\alpha f_1}{\gamma}, \frac{\beta f_1}{\gamma}) & V_2(\lambda, \mu) \\
\text{Case 3:} & V_1(\alpha, \beta) & E_2(\frac{\lambda f_2}{\nu}, \frac{\mu f_2}{\nu}) \\
\text{Case 4:} & V_1(\alpha, \beta) & V_2(\lambda, \mu)
\end{array}
$$

### 4.5.4 – Epipolar line calculation

### 4.5.5 – CASE 1: two epipoles

#### *a) theory*

Let $M_1(x_1, y_1, z_1)$ be a point in $P_1$. $E_1M_1$ defines an epipolar line in $P_1$, and the corresponding $E_2M_2$ defines the conjugate epipolar line in $P_2$. The plane $(E_1, \theta_1, \theta_2, E_2, M_1, M_2)$ contains the translation vector $t$ and $E_1M_1$. Its normal is $E_1M_1 \times t$. The normal of $P_2$ is $\theta_2 z$. Hence the intersection of the two planes is given by the vector:

$$\theta_2 z \times (E_1 M_1 \times t) = (\theta_2 z \cdot t) E_1 M_1 - (\theta_2 z \cdot E_1 M_1) t$$

and $E_2M_2$ is collinear to this vector. Suppose that in $\theta_1 xyz$, $E_1M_1$: $(x_1, y_1, 0)$. In terms of components in $\theta_2 xyz$:

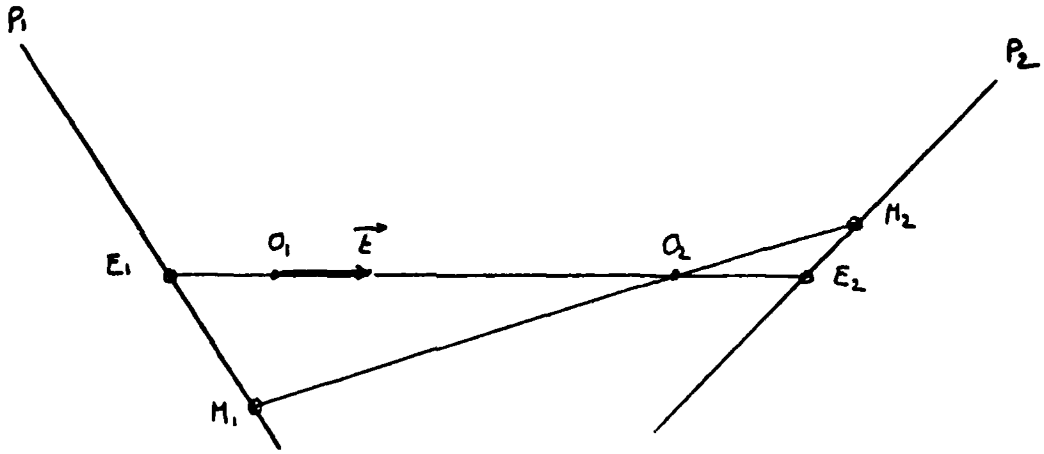
$$\theta_2 z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad t = \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix}, \quad E_1 M_1 = \begin{pmatrix} x'_1 \\ y'_1 \\ z'_1 \end{pmatrix} = \begin{pmatrix} r_{11}x_1 + r_{12}y_1 \\ r_{21}x_1 + r_{22}y_1 \\ r_{31}x_1 + r_{32}y_1 \end{pmatrix}$$

$$E_2 M_2 \text{ is collinear to } \begin{pmatrix} \nu x'_1 - \lambda z'_1 \\ \nu y'_1 - \mu z'_1 \end{pmatrix} = \begin{pmatrix} x_1(\nu r_{11} - \lambda r_{31}) + y_1(\nu r_{12} - \lambda r_{32}) \\ x_1(\nu r_{21} - \mu r_{31}) + y_1(\nu r_{22} - \mu r_{32}) \end{pmatrix}$$

If we let A be the matrix:

$$\begin{pmatrix} \nu r_{11} - \lambda r_{31} & \nu r_{12} - \lambda r_{32} \\ \nu r_{21} - \mu r_{31} & \nu r_{22} - \mu r_{32} \end{pmatrix}$$

Then we can write $E_2M_2 = AE_1M_1$ where $E_1M_1$ is in base $\theta_1xy$ and $E_2M_2$ is in base $\theta_2xy$.




*Case 1*
Figure 4-4

***b) algorithm***

Let $N_1$ be the number of epipolar lines that we want to determine. Each epipolar line is uniquely determined by the angle it makes with the $x$-axis. Let $\theta$ be this angle. Given $k$, the epipolar line number, $0 \leq k \leq n1 - 1$, how can we determine $\theta$? If $\theta_0$ and $\theta_1$ are the lower and upper limits between which $\theta$ is allowed to vary, and $\theta_2 = \frac{(\theta_1 - \theta_0)}{N_1}$, then we will choose the middle of each interval: $\theta = \theta_0 + (k + .5)\theta_2$ But what are $\theta_0$ and $\theta_1$? We have to distinguish between three cases:

- The epipole is in the picture (very unlikely). Then $\theta$ can vary between 0 and $2\pi$ radians;

- The epipole is outside the image: there is a minimum and maximum angle under which the image is seen from this point. If we choose these angles in $[0, 2\pi]$, then most of the time every $\theta \in [\theta_0, \theta_1]$ will define a valid epipolar line in $P_1$;
- The exception from above is when the epipole is left of the image but on a same vertical level: then $[\theta_0, \theta_1]$ cannot be connected and still included in $[0, 2\pi]$. In this case we will choose the angles in $[\frac{-\pi}{2}, \frac{\pi}{2}]$.

Then $L_1$ will be defined by the point $E_1$ and the vector $V_1(cos\theta, sin\theta)$, and $L_2$ is defined by $E_2$ and $V_2 = AV_1$.

## 4.5.6 – CASE 2: one epipole $E_1$

### *a) theory*

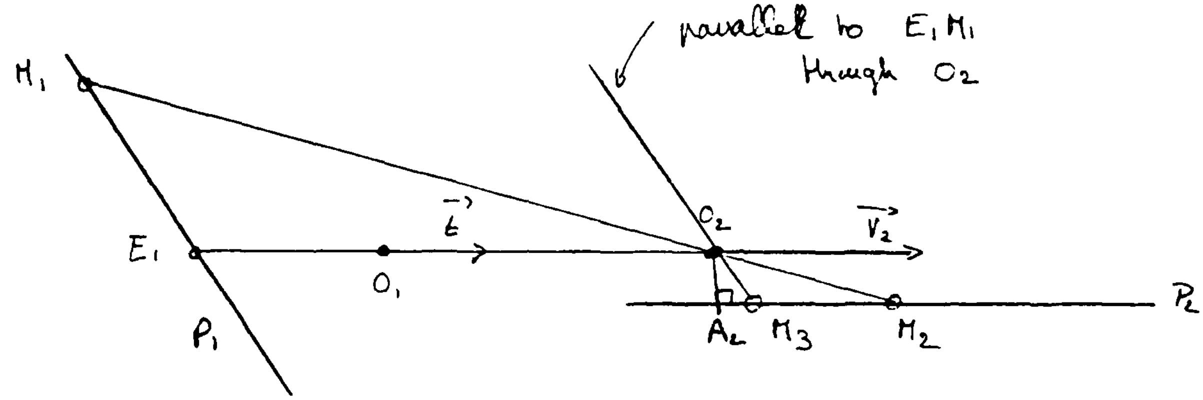
Given an epipolar line $\frac{L_1}{E_1M_1}$, we already know that the corresponding epipolar line $L_2$ in $P_2$ is collinear to the vector $t = V_2(\lambda, \mu)$. Hence we just need to find a point belonging to $L_2$. Clearly, $L_2$ is the intersection of $P_2$ with the plane $(E_1, M_1, \theta_2)$. Thus any line contained in this plane will intersect $P_2$ at a point contained in $L_2$. In particular, consider the parallel to $L_1$ driven through $\theta_2$. It intersects $P_2$, thus $L_2$, at $M_3$ such that, in base $\theta_2 xyz$:

$$E_1M_1 = \begin{pmatrix} r_{11}x_1 + r_{12}y_1 \\ r_{21}x_1 + r_{22}y_1 \\ r_{31}x_1 + r_{32}y_1 \end{pmatrix}, \quad \text{hence} \quad \theta_2 M_3 = \begin{pmatrix} f_2 \frac{(r_{11}x_1 + r_{12}y_1}{r_{31}x_1 + r_{32}y_1)} \\ f_2 \frac{(r_{21}x_1 + r_{22}y_1}{r_{31}x_1 + r_{32}y_1)} \\ f_2 \end{pmatrix}$$

### *b) algorithm*

In the same way as in case 1, we define $L_1(E_1, V_1)$ where $V_1(x_1 = cos\theta_1, x_2 = sin\theta_2)$. Then $L_2$ is defined by $(M_3, V_2)$, where the coordinates of $M_3$ are

$$\left( f_2 \frac{r_{11}x_1 + r_{12}y_1}{r_{31}x_1 + r_{32}y_1}, f_2 \frac{r_{21}x_1 + r_{22}y_1}{r_{31}x_1 + r_{32}y_1} \right)$$




*Case 2*
Figure 4-5

**4.5.7 – CASE 3: one epipole $E_2$**

***a) theory***

Let an epipolar line in $P_1$ be defined by the translation vector $t = V_1$ and by a point $M_1$ we pick. In $P_2$, $L_2$ goes through $E_2$ and is collinear to a vector $E_2M_2$, intersection of $P_2$ with the plane $(\theta_2, E_2, M_1)$. This plane is orthogonal to $\theta_2 M_1 \times t$ and $P_2$ is orthogonal to $\theta_2 z$. Hence the intersection is collinear to $\theta_2 z \times (\theta_2 M_1 \times t)$

$$\begin{aligned} \text{since } \theta_2 M_1 &= \theta_2\theta_1 + \theta_1 M_1 \\ &= kt + \theta_1 M_1, \\ \theta_2 M_1 \times t &= \theta_1 M_1 \times t, \\ \text{and } \theta_2 z \times (\theta_2 M_1 \times t) &= (\theta_2 z \cdot t)\theta_1 M_1 - (\theta_2 z \cdot \theta_1 M_1)t. \end{aligned}$$

Suppose $\theta_1 M_1 : (x_1, y_1, f_1)$ in $\theta_1 xyz$. Then in $\theta_2 xyz$:

$$\theta_2 z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad t = \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix}, \quad \theta_1 M_1 = \begin{pmatrix} x'_1 \\ y'_1 \\ z'_1 \end{pmatrix} = \begin{pmatrix} r_{11}x_1 + r_{12}y_1 + r_{13}f_1 \\ r_{21}x_1 + r_{22}y_1 + r_{23}f_1 \\ r_{31}x_1 + r_{32}y_1 + r_{33}f_1 \end{pmatrix}$$

$E_2 M_2$ is thus collinear to:

$$\begin{pmatrix} \nu x'_1 - \lambda z'_1 \\ \nu y'_1 - \mu z'_1 \end{pmatrix} = \begin{pmatrix} x_1(\nu r_{11} - \lambda r_{31}) + y_1(\nu r_{12} - \lambda r_{32}) + f_1(r_{13} - \lambda r_{33}) \\ x_1(\nu r_{21} - \mu r_{31}) + y_1(\nu r_{22} - \mu r_{32}) + f_1(\nu r_{23} - \mu r_{33}) \end{pmatrix}$$

Hence, if we let $A$ be the same matrix as in CASE 1 and OF3 be the offset matrix:

$$\begin{pmatrix} \nu r_{13} - \lambda r_{33} \\ \nu r_{23} - \mu r_{33} \end{pmatrix}$$

Then we have: $E_2 M_2 = A\theta_1 M_1 + OF3$

Where $\theta_1 M_1$ is in base $\theta_1 xy$ and $E_2 M_2$ is in base $\theta_2 xy$

***b) algorithm***

Now, how do we pick $M_1$ in the first place? We want a set of $N_1$ equally spaced epipolar lines, and it appears convenient to pick points on the axes. If the epipolar lines are more horizontal, or the image stretched in height, then we will pick $N_1$ equally spaced points on the vertical axis, suitably located to cover the entire image. If the epipolar lines are more vertical or the image more stretched in width, then we pick them on the horizontal axis. Let $L_x, L_y$ be the picture dimensions and $(V_x, V_y)$ the epipolar direction:

If $V_x L_y < V_y L_x$, and $V_y < 0$, we pick $y_1 = (L_x|\frac{V_y}{V_x}| + L_y)(\frac{K+0.5}{N_1})$

If $V_x L_y < V_y L_x$, and $V_y > 0$, we pick $y_1 = (L_x|\frac{V_y}{V_x}| + L_y)(\frac{K+0.5}{N_1}) - L_x\frac{V_y}{V_x}$

If $V_y L_x < V_x L_y$, and $V_y < 0$, we pick $x_1 = (L_y|\frac{V_x}{V_y}| + L_x)(\frac{K+0.5}{N_1})$

If $V_y L_x < V^x L_y$, and $V_y > 0$, we pick $x_1 = (L_y|\frac{V_x}{V_y}| + L_x)(\frac{K+0.5}{N_1}) - L_y\frac{V_x}{V_y}$

Then we proceed as indicated above:

$$L_1(M_1, V_1 = t) \text{ is matched with } L_2(E_2, V_2 = F_2 M_2).$$

### 4.5.8 – CASE 4: no epipoles

#### *a) theory*

Let an epipolar line in $P_1$ be defined by the translation vector $t = V_1$ and by a point $M_1$ we pick. In $P_2$, $L_2$ goes through the image of $M_1$, that is the extremity of a vector collinear to $\theta_1 M_1$ and whose third component is $f_2$. In base $\theta_1 xyz, \theta_1 M_1 : (x_1, y_1, f_1)$. In base $\theta_2 xyz$:

$$\theta_1 M_1 = \begin{pmatrix} r_{11}x_1 + r_{12}y_1 + r_{13}f_1 \\ r_{21}x_1 + r_{22}y_1 + r_{23}f_1 \\ r_{31}x_1 + r_{32}y_1 + r_{33}f_1 \end{pmatrix} \quad \theta_2 M_2 = \begin{pmatrix} f_2 \frac{r_{11}x_1 + r_{12}y_1 + r_{13}f_1}{r_{31}x_1 + r_{32}y_1 + r_{33}f_1} \\ f_2 \frac{r_{21}x_1 + r_{22}y_1 + r_{23}f_1}{r_{31}x_1 + r_{32}y_1 + r_{33}f_1} \\ f_2 \end{pmatrix}$$

#### *b) algorithm*

We pick points $E_1$ in the same manner as in case 3. Then we calculate $\theta_2 M_2$ as indicated above and we match $L_1(E_1, t)$ with $L_2(E_2, t)$.

## *4.6 Transform Parameters*

We want to output the minimum necessary parameters to do the following transformation:
given the coordinates of one point in one image, find its coordinates in epipolar space.

We treat the problem only in CASE 1, which is the most common case encountered. For this case, we will need:

- the coordinates of the epipoles;
- the number of epipolar lines;
- the minimum and maximum angle $\theta_0$ and $\theta_1$ under which picture 1 is viewed;
- the minimum distances $D_1 min$ and $D_2 min$ of the epipoles to the images;
- the matrix $R$, or rather its inverse $R^{-1}$.

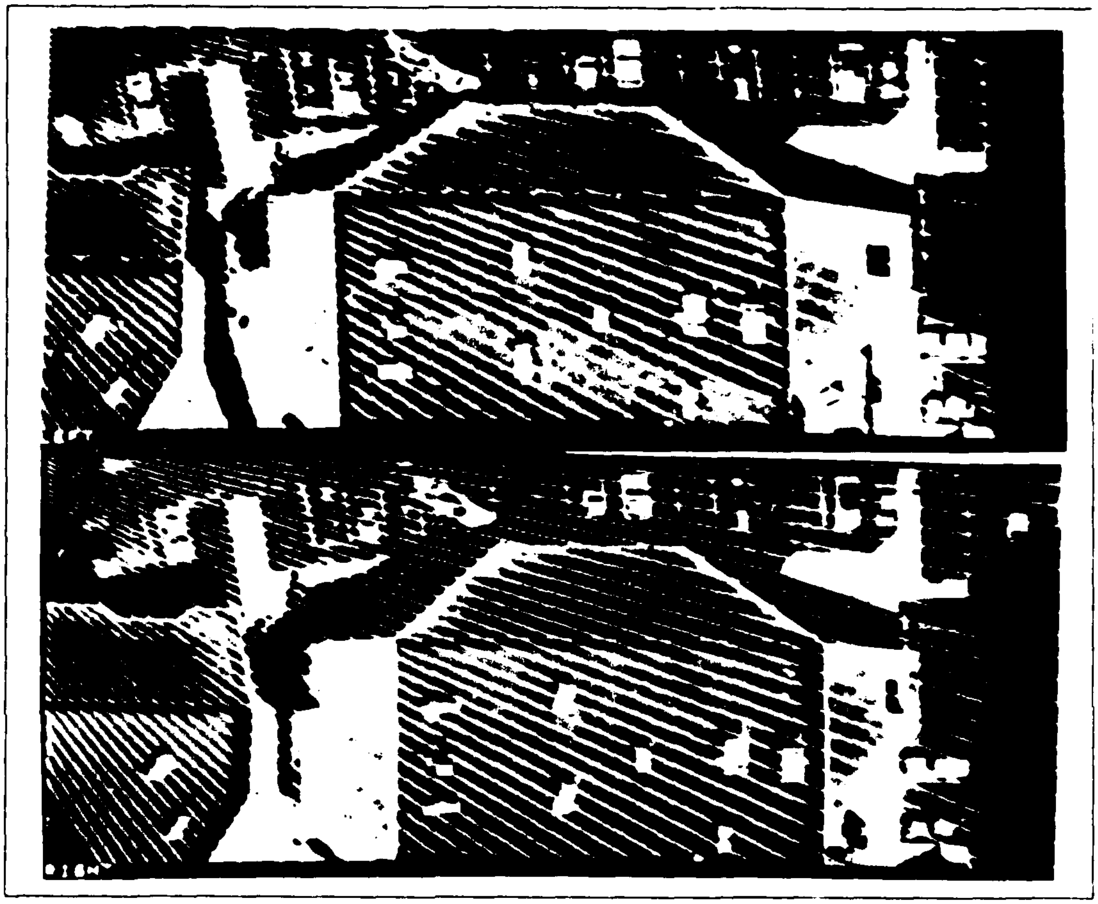
For example, the images of figure 5-6 have transform parameters as follows:

$$
\begin{aligned}
Case &= 1 \\
N &= 450 \\
E_{11} &= 356755.3 \\
E_{12} &= 166509.8 \\
E_{21} &= -7765.348 \\
E_{22} &= -3509.268 \\
\theta_0 &= 3.578705 \\
\theta_1 &= 3.577792 \\
D_1 min &= 393249.0 \\
D_2 min &= 8521.478 \\
R_{11}^{-1} &= 430.7463 \\
R_{12}^{-1} &= 2375.517 \\
R_{21}^{-1} &= 215.9756 \\
R_{22}^{-1} &= 1075.811
\end{aligned}
$$

## 4.7 Example of Epipolar Registration and Transformation

Figure 4-6 shows a stereo pair of a building complex. Figure 4-7 has this pair superpositioned with a set of corresponding epipolar lines. Figure 4-8 shows the imagery transformed such that epipolar lines are horizontal in the image, and conjugate epipolar lines have the same row coordinate.

*Sacramento Building Image Pair*
Figure 4-6



*Epipolar Lines in this Imagery*
Figure 4-7

Transformed Imagery
Figure 4-8

# 5: Analysis of Automated Stereo Mapping

## *5.1 Background*

Results from our laboratory over the past few years [Quam 1971, Hannah 1974, Moravec 1980, Gennery 1980, Arnold 1980, Baker 1981, Arnold 1983], have demonstrated the possibilities of both area-based and feature-based stereo matching.

***Area-based*** stereo matching uses windowing mechanisms to isolate parts of two images for cross-correlation. ***Feature-based*** stereo matching uses two-dimensional convolution operators (and perhaps grouping operators) to reduce an image to a depiction of its intensity boundaries, which can then be put into correspondence. Area-based cross-correlation techniques require distinctive texture within the area of correlation for successful operation. They break track:

- where there are ambiguous textures or featureless areas (roofs, sand and concrete);
- where the correlation area crosses surface discontinuities (at occlusions such as buildings, or thin objects (poles));
- where depth is ill-defined (such as through trees).

In general, these systems break track where there is no local correlation (zero signal, or where two images do not correspond) or where the correlation is ambiguous (where the signal is repetitive). The systems must be started manually and corrected when they break track.

Demands of mapping in cultural sites and in locales with surface discontinuity and ambiguous or non-existent texture make it essential that, if area-based analysis is to be done, it be done in conjunction with feature-based analysis. ***Feature***-based analysis provides a solution to many of the problems of correlation. Principal among its advantages is that it operates on the most discriminable parts of an image: places that are distinctive in their intensity variation, and where localization is greatest. These are typically the boundaries between objects or between details on objects, or between objects and their backgrounds. The important point is that the features being put into correspondence for depth estimates are the boundaries of objects: area-based analysis is at its worst at object boundaries, yet determining boundaries can be said to be the most important part of mapping in 3-space.

Some other advantages of feature-based analysis are:

a) ***reduced combinatorics*** — there are fewer features than pixels,

b) ***greater accuracy*** — features can, in general, be positioned to sub-pixel precision, while area positioning precision is inversely proportional to window size, and considerably poorer, and

c) ***more realistic invariance assumptions*** — area-based analysis presupposes that the ***photometric*** properties of a scene are invariant to viewing position, while feature-based analysis works with the assumption that it is the ***geometric*** properties that are invariant to viewing position.

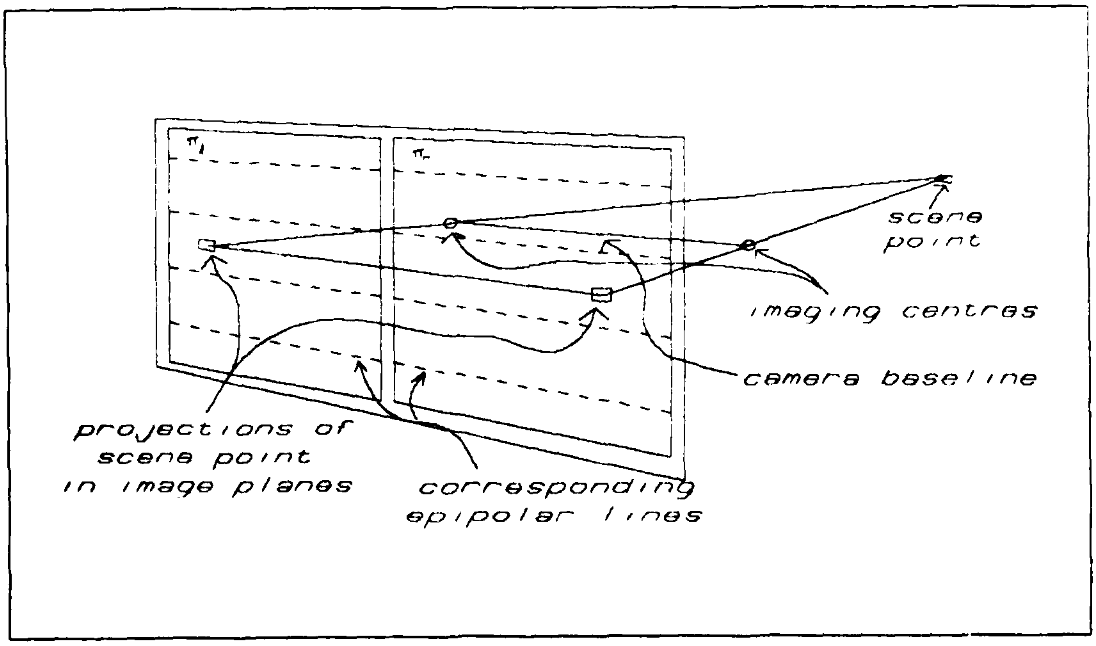
The Baker system [Baker 1981] is the only current system that mixes these two matching modalities. We undertook in this postdoctoral work to demonstrate the capabilities of this system applied to new cultural imagery.

## *5.2 Baker System: before*

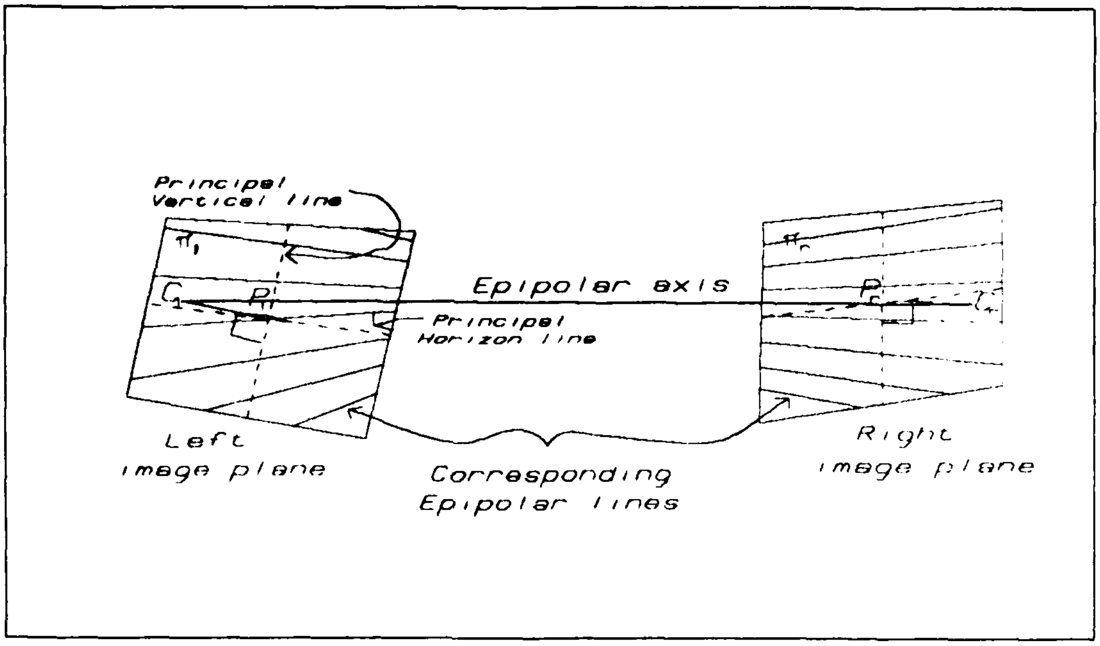
In the Baker system, image ***edges*** are the features *considered in stereo matching.* These edges are located at positions in the image where a change in sign of second difference in intensity occurs. In the initial case a particular operator (one that was 1 by 7 pixels in size) measured the directional first difference in intensity at each pixel. Second differences were computed from these, and changes in sign of these second differences were used to interpolate zero crossings (*i.e.* peaks in first difference). Certain local properties other than ***position*** were measured and associated with each edge — ***contrast, orientation,*** and ***intensity to either side*** — and ***links*** were kept to nearest neighbors above, below, and to the sides. It is these properties that define an edge and provide the basis for the correspondence process.

The correspondence is a search for ***edge*** matches across images. With no prior knowledge of the viewing situation, one could have any edge in one image matching any edge in the other. The combinatorics of a naive matching strategy clearly could be enormous. A lot of the analysis of a matching strategy goes into bounding this combinatorics and constraining the search for edge correspondences.

One of the principal constraints used in stereo analysis may be determined from a knowledge of the relative attitudes of the cameras. If two equalivalent cameras are arranged with axes parallel, as shown in Figure 5-1, then they can be conceived of as sharing a single common image plane. Any point in the scene will project to two points on that joint image plane (one through each of the two lens centers), the connection of which will produce a line parallel to the baseline between the cameras. Thus corresponding edges in the two images must lie along the same line in the joint image plane. This line is termed an ***epipolar line*** (see [Hallert 1960], and section 4). If the baseline between the two cameras happens to be parallel to the scanning axis of the cameras, then the correspondence only need consider edges lying along matched lines parallel to that axis in the two images. These lines are termed ***conjugate***. Figure 5-1 indicates this camera geometry a geometry which produces ***collinear epipolar lines***. The algorithm described here assumed the stereo pair to have such a geometry, and if this was not the case then the appropriate transformation of the two images had to be made before any processing could be done. A less restrictive solution would be to have the correspondence process informed of the camera geometries, and have it solve for the more general epipolar situation as shown in figure 5-2. Incorporation of this capability is one of the enhancements we have made to the system over the past few months. Section 4 details the analysis for this solution.




Specialized Epipolar Geometry
Figure 5-1




*General Epipolar geometry*
Figure 5-2

The advantage of processing along epipolar lines is that the search for correspondence may be constrained to one dimension, along a single line-pair. It is then feasible to apply an efficient optimization process to the matching problem. The Viterbi algorithm [Forney 1973] is the optimization process used in this system. It is a dynamic programming technique used extensively in speech processing, and first used in vision research in some recent work at Control Data Corporation [Henderson 1979]. An earlier use of a dynamic programming technique for stereo matching is documented in [Gimel'farb 1972]. [Baker 1981] describes in detail the use of the Viterbi algorithm for stereo correspondence.

Other geometric and photometric constraints may be introduced both in limiting search and in bounding parameters for the evaluation function. [Arnold 1980] and [Baker 1981] provide discussions of these constraints in determining edge correspondences.

## 5.3 Baker System: enhancements

As stated, we undertook in this postdoctoral work to demonstrate the capabilities of the Baker system [Baker 1981] applied to some cultural scenes. Before carrying out these analyses we:

a) enhanced the system with a capability to work with a better edge operator [Marimont 1982];

b) enabled it to process images that are not graced with collinear epipolar geometry (*i.e.* most images);

c) introduced an additional correspondence measure - edge extent.

To implement these enhancements required substantial redesign of the system, and redesign cycles with the Marimont process. Chosing useable data also presented difficulties, as the only imagery available was not of the correct geometry (see below). The two image pairs initially chosen (the Sacramento apartment complex and a section of some imagery of Moffett Field) proved, on closer examination, to require quite complex transformation, and could not be easily adjusted for epipolar processing. Obtaining useable data then tied the progress of this part of the demonstration to the progress of developing an image registration system.

In general, to bring imagery data into a properly transformed state could proceed in one of two ways:

- one could determine the transforms and then modify the imagery, producing an image pair having collinear epipolar geometry;

or

- one could determine the transforms, and modify the output of an edge operator process that functions over the original imagery.

The latter is by far the superior approach, as it avoids resampling the image. This approach necessitates incorporating the transform computation into the stereo system, to follow edge finding and precede edge matching.

The second part of the stereo system's analysis is an intensity correlation process. This operates along epipolar lines as well, and clearly requires intensity information to be accessable along epipolar lines. One solution to this would be to take the original image pair and have the correlator rotate and change shape, size, and orientation as it moves around the image; this is an awkward and probably unnecessary complication. An alternative would be to access the transformed images, sampled as accurately as possible, and do the correlation in the rectangular space defined by collinear epipolar lines. The argument from edge accuracy indicated that transforming edges rather then resampling the image was the way to go; this argument from intensity correlation suggests that the resampled image can be useful.

Another implementation detail supported this use of both transformed edges and transformed imagery: it was found that the intensity information available from the Marimont process had too small a basis for useful correlation, and in fact, for transformed edges, had little relevance for the matching (it being measured not along epipolar lines). The transformed image had to be referenced again by the system to obtain more significant intensity estimates oriented along epipolar lines, and working with the image in epipolar space facilitated this.

The philosophy of the stereo matching process here had been to use edge analysis for, among other things, its higher accuracy, and to use intensity analysis for the continuity it provides. To be consistent with this, we wanted to have the highest possible accuracy for edges in epipolar space, and if sacrifice be needed for simplicity, to do it where it least degraded the analysis - in the intensity correlation. It is clear that transformed edges give higher accuracy than edges from transformed images (detectability might not change much, but localization is significantly reduced); and important simplifications could be obtained for little loss by doing the intensity correlation over the resampled image pair. This meant changes in our plans for the registration system: it had to produce not just transform information, but transformed images as well. Both forms are made available as output from the registration program described in section 4, and the enhanced Baker system uses them both. (However, the original edge finder of the stereo system has not been modified to use the transform information, and must work (at present) on the transformed images - with the limitations that brings.)

## *5.4 The Marimont Edge Operator*

The Marimont edge operator has greater detection and reliability than the original Baker edge operator, and similar localization; earlier examples of its processing convinced us that its output would improve the quality of our stereo reconstruction. Its ability to track along zero signal areas in following zero-crossing edges leads to more coherent image descriptions. [Marimont 1982] provides details of the operator's functioning. Roughly, it works by convolving an $m \times m$ lateral inhibition function of $n \times n$ central window with an image. Zero crossings in this resultant image then indicate edges, and the edge position is determined by interpolating over the lateral inhibition surface. Section 7 details the content and format of the edge files produced by this process.

A few unanticipated problems became apparent once work with the edges was begun. One point, noted above, was that the intensity information stored at an edge (its left and right boundary values) had quite small support (a single pixel). This is in contrast with the original operator which interpolated for these values in an area 3 pixels wide and removed one pixel from the determined edge position. Another problem was that the edge connectivity produced by the Marimont system can be misleading, as figure 5-3 shows. Intensity significance was improved by sampling along epipolar lines in the transformed images. The connectivity problem has not been looked at yet.

Good connectivity is inherently difficult to achieve with zero crossing operators. Refinements to the process are being considered.

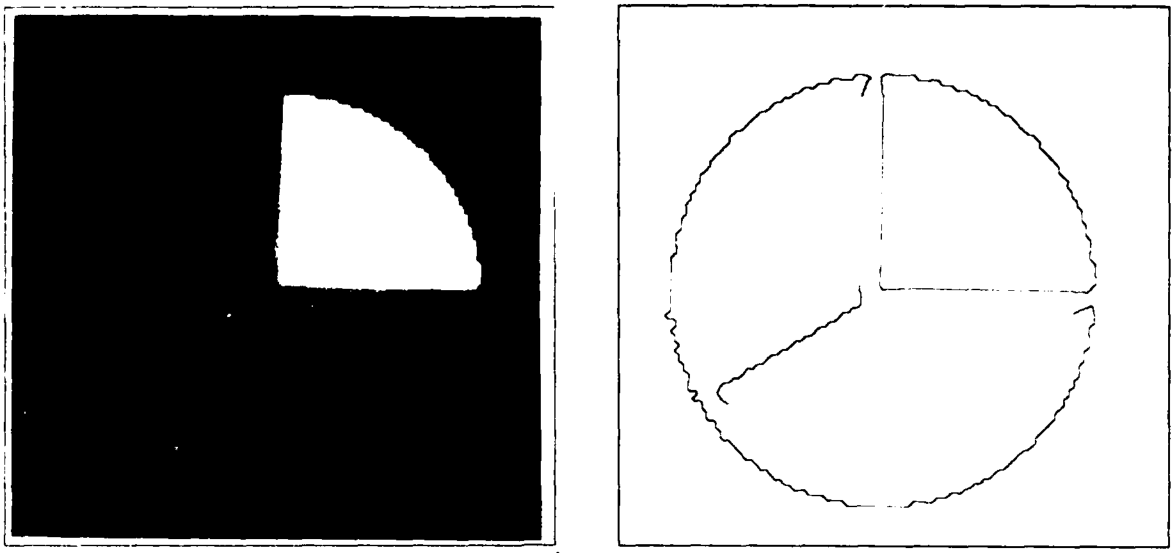


*Image*
Figure 5-3

The introduction of edge extent as a parameter in the dynamic programming solution was an obvious fallout from using the Marimont edges. Edges are output by that process as strings, with 2-connectedness. The maximum and minimum of some string, in transform space, is a measure of its (epipolar) extent. Prior to the use of this information the only way that global continuity entered the analysis was through a consistency enforcement relaxation process which ensured that edges connected in one view were interpreted as continuous in 3-space; all matching measures were quite local. With the modified approach, the correspondence measure is a function of (among other, more statistically based parameters) the ratio of edge extents. In particular, the likelihood of edge element $a$ in the left image matching edge element $b$ in the right image depends on the product of the ratios of the two upper extents (up from the edge elements) and the two lower extents (down from the two edge elements).

When image testing began with all of the above accomplished, another problem became apparent: the stereo system, bound into a machine architecture with a maximum of 256K words of memory, and always tightly wedged anyway, had grown with these changes to the point that only small portions of images could be worked on at once. Thus came to exist a windowing mechanism within the edge finding/loading and stereo matching processes.

## 5.5 Test Imagery

Our testing has been progressing on several sets of imagery: a synthetic image pair from Control Data Corporation, an aerial scene from the Engineering Topographic Laboratory, and a building scene of Sacramento. The latter imagery is the most relevant to the current postdoctoral contract, and we will be demonstrating the mapping system on this data. This imagery may be seen in figures 4-6 and 4-8.

# 6: Bitpad Instructions

## *6.1 Instructions for the Digitizing Tablet*

### 6.1.1 – Introduction

The bitpad program is written in PASCAL to be run on a DEC KL-10 processor under WAITS at the Computer Science Department of Stanford University. It has an assembly code driver to handle input from the tablet over a normal RS-232 port (listing included also), and requires a graphics package available locally. Replacing this package for an intended use away from SAIL WAITS should be quite straightforward.

Locally, the bitpad may be run by typing DO TAB[S,HHB]. Figure 6 shows the digitizing tablet in use.

The initial preamble of questions deals with graphics options and line characteristics. You will usually want some sort of graphics output, so answer yes to the first question ("Want graphics output (y/n)?:"). Graphics choices are DD or GOD (for later display). Say y to either or both of the questions about this. DD graphics go the Data Disc display while for viewing while you construct an edge file, while GOD graphics are for later Graphics Output Device device independent display. If you want little boxes drawn around the endpoints of lines, then asnwer y to ("Want boxes draw around vertices (y/n)?:"). Box size refers to the size of the box to be drawn around vertices. Small is better for entry, large makes it easier to distinguish vertices from line nodes if you are trying to do deletions.

On the first digitizing pass over an image you will start directly from the bitpad; later sessions will involve adding to or editing an existing edge file. To enter an old file (or several old files), say Y to the question ("Want to read in a file (y/n)?:"). This reads it in and displays it. For the moment there's a problem with reading strings into PASCAL, so it tries to read NEW.VRT[S,HHB]. If this file doesn't exist, it will give you the chance to enter another filename.. don't panic, just type in the new name.

When that file has been read in and displayed, it lets you choose another (ad infinitum). End with an answer of N (for NO) to the above question.

### 6.1.2 – Setting screen coordinates

Whether you've read in a file, or not, you now set up the bitpad to enter data. It asks for the TTYline of the tablet (50 is the current line number of the bitpad plug). You then define the screen area on the bitpad: select the four corners with the crosshairs (lower left, upper left, upper right, lower right) with any button, and confirm your selection by pushing button '2' ('8' says no, try for four more points).

#### 6.1.3 – Adding segments and vertices

A 'segment' is made up of a start vertex, any number (even 0) of intermediate points, and an end vertex. Vertices may belong to any number of 'segments'. Adding vertices/points/segments is as follows:

Button: '1' defines a new vertex at the current crosshairs position, either starting a segment with it, or ending one there. '2' locates the nearest vertex (created by '1'), and begins or ends a segment there. '4' says add this point to the current segment list. '8' is a break. It lifts the pen. If it is followed by a '1' or '2', then that is all it does. If followed by a '4', it puts you in the delete cycle. If followed by another '8', you exit everything.

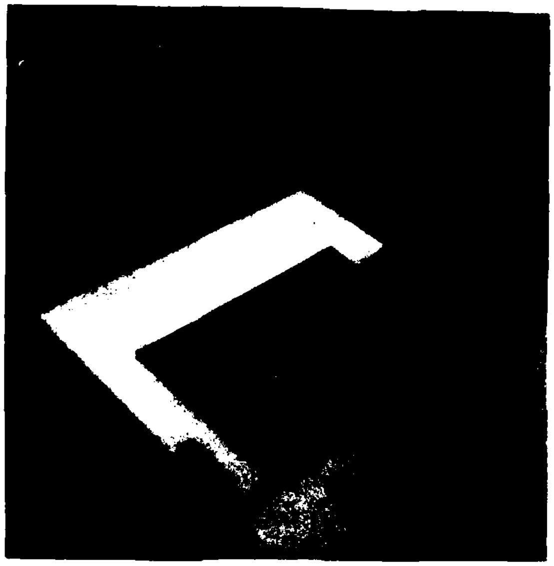
#### 6.1.4 – Editing segments

You may enter the editor to delete segments at any point. You do this by keying '8' then '4' in the above entry loop.

To delete segments, first choose a vertex attached to the segment with '1' button. Then select either the other vertex (if there is no ambiguity) with a '1', or a point somewhere on the segment with a '4'. You then see it erased on the screen, and you push '2' to confirm that that's the one you want deleted, or '8' to say that its not the one you want deleted. Leave the delete cycle by keying '8'.

If you wish to add more to the file, run it again, and this time read in the VRT file produced on this previous pass.

At the moment, the GOD file is the sum of everything drawn on the screen during the run. To get just the current description, you have to pass the file once more through the program; making no changes, just reading it in and writing it out. Because of this, I recommend NOT producing a GOD file on the $1^{st}$ through $n - 1^{st}$ creation passes, saving it for the last run, when nothing will be altered.



*The Digitizing Tablet*
Figure 6-1

# 7: Marimont Edge File Format

This section describes the format of the edge files created by the Marimont edge process.

## *7.1 Header*

The file is binary, and begins with an 128-word header, followed by a list of linked edge lists (ledgels); that is, after the header, there are a number of items called ledgels, each of which is a list of edges.

The format of the header is as follows:

| word | interpretation |
|---|---|
| 0 | unused; ignore |
| 1 | unused; ignore |
| 2 | bits per pixel in input picture |
| 3 | bias of laterally inhibited picture |
| 4 | "nil-strength" flag |
| 5 | "end-of-list" flag |
| 6 | first row of picture window |
| 7 | first col of picture window |
| 8 | last row of picture window |
| 9 | last col of picture window |
| 10 | 1 if contours of positive regions, -1 if those of negative regions. |
| 11 | format number of edge file (format 3 currently); |
| 12-127 | unused; ignore |

## *7.2 Ledgels*

Next come one or more ledgels, followed by a word containing the end of list flag (word 5 of the header above). Each ledgel has a two word header, a list of one or more edges, and a word after the last edge containing the end of list flag. Note that this word flags the ends of two kinds of list: the list of ledgels, and the list of edges that each ledgel contains.

Each ledgel has the following format:

| word | interpretation |
|---|---|
| 0 | 1 if the ledgel is closed, *i.e.* the last edge is linked (conceptually) to the first, else 0. |
| 1 | number of left turns less the number of right turns in the ledgel, as one traverses the ledgel from head to tail. |
| 2 | beginning of this ledgel's list of edges |

. [one or more edges]

.

last end-of-list flag

## *7.3 Edgels*

Each ledgel has a list of one or more edges. There are two kinds of edges, corresponding to "significant" and "insignificant" zero crossings in the laterally inhibited image. A significant zero crossing is a transition of the laterally inhibited signal from positive to negative. An insignificant zero crossing is a transition of the laterally inhibited signal from positive or negative (depending on the sign of the region whose encircling contours are being detected, see word 10 of the header above) to zero, and as such does not correspond to a likely edge in the original image. Their exclusion from linked edge lists, however, seems to destroy much useful connectivity. So, we include them as "links" and assume that one between two significant zero crossings is not a significant break in the edge. But at insignificant zero crossings we do not estimate the strength of the edge or image intensities to either side as we do for significant zero crossings. Since less information is thus associated with insignificant zero crossings, we use two edge formats to save space.

The format for an edge is as follows:

| word | interpretation |
|---|---|
| 0 | interpolated row coordinate of edge point (floating-point) |
| 1 | interpolated column coordinate of edge point (floating-point) |
| 2 | integer equal to 0 if edge is horizontal, 1 if vertical |
| 3 | two possible interpretations: if when interpreted as an integer, it equals the nil-strength flag, then this is an insignificant edge point, the edge strength is undefined, and this is the last field in the edge. Otherwise, this edge is significant, this field should be interpreted as a floating-point estimate of the edge strength, and the next two words are fields belonging to this edge. all edge files as of this writing were created by encircling positive regions (see word 10 of the header), and in that case the edge strength is always positive, and the brighter side of the edge is towards the left as one moves from the head of the ledgel to the tail. |
| 4 | (significant edges only!) estimate of original image intensity to the left |
| 5 | (significant edges only!) estimate of original image intensity to the right |

## *7.4 Clarifications*

The edge location coordinates are in picture coordinates, i.e. with respect to the original picture, NOT w.r.t. the window in which they were detected. Integral locations correspond to pixel centers, not boundaries, and the topmost, leftmost pixel center is taken to be $(0, 0)$, which means that an $m \times n$ picture has its upper left corner at $(-0.5, -0.5)$, and its lower right hand corner at $(m - 0.5, n - 0.5)$ (since the indices of the last row, col are $(m - 1, n - 1)$).

For the most part, edges are detected between horizontally or vertically adjacent pixels, which is why there are only two possible orientations for edges. An edge is detected between two such adjacent pixels if the laterally inhibited signal is positive at one and negative at the other. The edge location is estimated by the linearly interpolated zero crossing between these two values. Edge strength is the difference between laterally inhibited intensities one pixel to either side of estimated zero crossing (in the directions normal to that of the zero crossing); linear interpolation is used to estimate each of these intensities, since the points at which they're measured usually fall between pixels. The strength is scaled down to correspond to the difference between adjacent pixels (i.e. the strength measure is laterally inhibited contrast per pixel unit measure). Image intensities are estimated at these same points, i.e. one pixel to either side of the interpolated zero crossing, in an analogous way, by linearly interpolating between each pair of image intensities. However, both image intensities are retained.

The reason for the "for the most part" in the first line of the paragraph above is that "extended" zero crossings are detected in the current implementation. An extended zero crossing is three adjoining pixels in a row or column with one end pixel positive, one end pixel negative, and the middle pixel zero (values refer to the laterally inhibited signal). The estimates of edge location and strength and left and right image intensities are made just as in the above: the location of the edge is an interpolated zero-crossing, the edge strength is the (scaled) difference in (interpolated) laterally inhibited pixel values one pixel to either side of the estimated location, and the left and right image intensities are interpolated image intensities one pixel to either side of the estimated location.

# 8: Conclusions

## 8.1 Summary

A principal research interest of our group is in developing a rule-based advanced automated stereo mapping system to function within ACRONYM [Brooks 1981]. Current mapping techniques ignore much of the information available from inference on single views of a scene. This information can be useful for three-dimensional surface interpretation, and also provides extra parameters for stereo matching (*i.e.* surface orientation, occlusion cues). Our research effort is directed at establishing such monocular inference rules in a rule-base for stereo mapping.

In deriving these rules, we perform analysis of both hand extracted and automatically produced edge descriptions. A facility has been developed under the current postdoctoral contract for this manual edge extraction from hardcopy imagery. We have studied rule synthesis for several cases, including that of orthogonal trihedral vertices - features that dominate cultural scenes. This research is very promising, and has shown the utility of the rule-based approach to surface inference from monocular information. We will be continuing our research in rule synthesis under other funding.

Camera solving provides powerful constraint on the correspondence problem in stereo matching. We have developed a facility under this contract for interactively registering images, determining the parameters for transforming them (or their edge descriptions) into collinear epipolar space, and performing the actual image transformation. This determination is crucial to a mapping process. Incorporating an automated module to provide data for the camera solving is a very important next step.

We have experimented with an existing stereo mapping process, enhancing its flexibility with respect to image format and with respect to edge operator format, and have been preparing example outputs of its processing on new imagery. Our intent with this effort has been to show the capabilities of a local matching process and to assess its applicability to the planned rule-based system.

## 8.2 Demonstration

We will be demonstrating the following for RADC later in May:

a) an interactive digitizing test facility;

b) the use of image edge descriptions produced using this facility and from an automated process [Marimont 1982] in synthesizing rules for stereo matching;

c) examination of Orthogonal Trihedral Vertex (OTV) inference rules;

d) development of a system for registration of image pairs;

e) analysis of stereo imagery with an automated stereo process.

Our demonstration will also present related equipment and system facilities that we use in support of this and other vision research.

# 9: Digitizing Program

See sections 2 and 6 for documentation on using this program.

---

```
{$D+}{$H:20000}
Program Tablet(tty,vertfile);


CONST
    CHOP=100000;
    PI = 3.14159265;
    DDX = 432.0; DDY = 432.0; GDD = 512.0;
    lfx = 40.0;
    maxdifference=0.05;  { one twentieth of screen away }

{ to hold the lines associated with a vertex. incoming lines are negative. }
TYPE
  LINESPTR=↑linestype;
     linestype=record
                  line:integer;
                  nextline:linesptr;
                end;


{ to hold the vertices with their associated coordinates and lines }
  VERTLISTPTR=↑vertlist;
     vertlist=record
                 vertex:integer;
                 x,y:real;
                 header:linesptr;
                 nextvert:vertlistptr;
               end;

{ to hold the points associated with a line }
  POINTPTR=↑point;
     point=record
                 xp,yp:real;
                 nextpt:pointptr
          end;

{ to hold the lines with their associated vertices and points }
  LINELISTPTR=↑linelist;
     linelist=record
                 linenum,initvert,finalvert:integer;
                 header:pointptr;
                 nextline:linelistptr
               end;


     TEMPARRAY=array[1..20] of integer;
     STRNG=packed array[1:30] of char;

INTFILE= FILE OF INTEGER;               { For graphics }

Var  savex,savey,X1,Y1,X2,Y2,X3,Y3,X4,Y4,MINX,MINY:real;
     signold,oldvert,linenum,pich,picw,vertcount,linecount,verta,vertb:integer;
     dx,dy,gx,gy,lstx,lsty,ttyline,boxsize:integer;      { for drawing (x,y) }
     theta,REALX,REALY:real;
     B1,B2,answer,dumchar:char;
     TABLETIO,jump,finished,ddgraph,LAST4,BOXFLAG,boxgod,
        toconfirm,needconfirm,DDGRAFX,GODGRAFX,T,readin,batch:boolean;
```

```
    vertfile,infile:text;
    filename:strng;
    vl,startvert,tailvert,
       tptr,vptr:vertlistptr;                   { vertex list }
    newlptr,lptr,l,
       scanptr,lstscanptr:linesptr;             { lines of vertex list }
    ll,startline,tailline,
       sptr,savsptr,lstsptr:linelistptr;        { line list }
    firstp,p,oldp,pptr:pointptr;                { points in line list }


    godfile: INTFILE;            { GOD files for plotting }
    SWITCH: RECORD               { Little hack for putting reals into integers }
       CASE FOO:BOOLEAN OF       { for GOD files }
               TRUE: (I:INTEGER);
               FALSE: (R:REAL)
           END;

{ external DD graphics procedures in GPAS.REL[TST,AAM]  }
procedure ginit; extern;         { init }
procedure scrset; extern;        { clear screen }
procedure width(w:integer); extern; { no-op }
procedure drken; extern;         { make line dark }
procedure liten; extern;         { make line light }
procedure inven; extern;         { swap above }
procedure move(x,y:integer); extern;    { invisible line to x,y }
procedure draw(x,y:integer); extern;    { visible line to x,y }
procedure dpyup; extern;         { draw to screen }
procedure inchar(var key,ctlkey,metakey:integer); extern;
procedure bitini(line:integer); extern; { init tty for input }
procedure bitpad(var resul1,resul2,resul3:integer); extern;     { tablet input }
function ttytab(var  esul1,resul2,resul3:integer):boolean; extern; {tablet or tty test }
procedure ppset; extern;         { set page printer }
procedure ppdone; extern;        { release page printer }




-----------------------------------------------------------------------------

{ READTABLET DISTANCE INITTEMPARRAY GRAPHICS (DD AND GOD) }

Procedure READTABLET(var B:char;var X,Y:real);
var T,IX,IY:integer;TA:real;

{ This procedure reads in data from either the terminal or the tablet.
   The data consists of which button was pressed, and the coordinates
   of the point at which it was pressed. The boolean TABLETIO indicates
   whether the procedure is reading from the tablet or from the terminal. }

    Begin
        if TABLETIO then
          begin
            bitpad(T,IX,IY);
{           IF TTYTAB(T,IX,IY) THEN WRITELN(TTY,'tty activity');}
{         writeln(tty,t:0,' ',ix:0,' ',iy:0); }
            B:=chr(48+T);X:=IX;Y:=IY;
          end
        else
          begin
            readln(tty);
            read(tty,B);read(tty,X);read(tty,Y);
          end;
    End;

  PROCEDURE GETCOORDS;
      Begin
        READTABLET(31,X1,Y1);
```

```
        REALX := (X1-MINX)/picw; REALY := (Y1-MINY)/pich;
        IF REALX < 0.0 THEN REALX:=0.0
           ELSE IF REALX > 1.0 THEN REALX:=1.0;
        IF REALY < 0.0 THEN REALY:=0.0
           ELSE IF REALY > 1.0 THEN REALY:=1.0;
{       writeln(tty,'B1=',B1, ', X1=',X1:0,'(',REALX:5:3,'), Y1=', Y1:0,'(',REALY:5:3,')'); }
      End;


FUNCTION ASK(N:integer): BOOLEAN;
VAR goodans:BOOLEAN;
  Begin { ASK }
    goodans:=FALSE;
    Repeat
        case N of
          1:WRITE(TTY,'Want Tablet IO (y/n)?:');
          2:WRITE(TTY,'Want graphics output (y/n)?:');
          3:WRITE(TTY,'  to the DD (y/n)?:');
          4:WRITE(TTY,'  to the GODFILE (y/n)?:');
          5:WRITE(TTY,'Want to read in a file (y/n)?:');
          6:WRITE(TTY,'Happy with this frame (y/n)?:');
          7:WRITE(TTY,'Want to DELETE segments (y/n)?:');
          8:WRITE(TTY,'Do another deletion (y/n)?:');
          9:WRITE(TTY,'Is this the line to be deleted (y/n)?:');
         10:WRITE(TTY,'  Want boxes draw around vertices (y/n)?:');
         11:WRITE(TTY,'  Want frame drawn in GOD file (y/n)?:');
         12:WRITE(TTY,'  Want to add to file from TABLET (y/n)?:');
        End;
        readln(tty);read(tty,answer); if(answer = 'y') or (answer = 'Y') then
                Begin ASK:=TRUE;goodans:=TRUE; End
           else Begin ASK:=FALSE;goodans:=TRUE; End;
      UNTIL goodans;
  End; { ASK }

Function DISTANCE(X1, Y1, X2, Y2: Real):Real; {Finds the distance bet. 2 pts. }
    begin
       distance:= Sqrt (Sqr (X2 - X1) + Sqr (Y2 - Y1));
    end;


Procedure INITTEMPARRAY(var a:temparray);
var i:integer;
    begin
       for i:= 1 to 20 do
          a[i]:=0;
    end;

Function MAX(a,b:integer): INTEGER;
    Begin
        IF A > B THEN MAX := A ELSE MAX := B;
    End;

{ Routines to make GOD files }

PROCEDURE OUT(N:INTEGER);
    BEGIN
        GODFILE↑ := N; PUT(GODFILE)
    END;

PROCEDURE OUTREAL(V:REAL);
    BEGIN
        SWITCH.R := V;
        OUT(SWITCH.I)
    END;

PROCEDURE PLOTLINE(X1,Y1,X2,Y2:REAL);
    BEGIN
        OUT(7);         { LINE }
        OUTREAL(X1); OUTREAL(Y1);
```

```
        OUTREAL(X2); OUTREAL(Y2);
        OUT(0)          { THICK }
    END;

{ Procedures to display graphics on the DD terminal. }


Procedure BOX;
  Begin { BOX }
    IF BOXFLAG THEN
      Begin
        IF DDGRAFX THEN
            Begin
                MOVE(dx-boxsize,dy-boxsize);DRAW(dx-boxsize,dy+boxsize);
                DRAW(dx+boxsize,dy+boxsize);Draw(dx+boxsize,dy-boxsize);
                Draw(dx-boxsize,dy-boxsize);MOVE(dx,dy);
                IF NOT(READIN) THEN DPYUP;
            End;
        IF GODGRAFX THEN
            Begin
                PLOTLINE(GX-boxsize,GY-boxsize,GX-boxsize,GY+boxsize);
                PLOTLINE(GX-boxsize,GY+boxsize,GX+boxsize,GY+boxsize);
                PLOTLINE(GX+boxsize,GY+boxsize,GX+boxsize,GY-boxsize);
                PLOTLINE(GX+boxsize,GY-boxsize,GX-boxsize,GY-boxsize);
            End;
      End;
  End; { /OX }

procedure DARKDRAW;
    Begin
        IF GODGRAFX THEN
                Begin
                    gx:=round(realx*GDD);
                    gy:= round(realy*GDD);
                End;
        IF DDGRAFX THEN
                Begin
                    dx := round(realx*DDX+lfx);
                    dy := round((1.0-realy)*DDY);
                    MOVE(dx,dy);
                    IF NOT(READIN) THEN DPYUP;
                End;    { move if DD, nothing if GOD }
    End;

procedure LIGHTDRAW;
    Begin
        IF GODGRAFX THEN
                Begin
                    lstx:=gx; lsty:=gy;
                    gx:=round(realx*GDD);
                    gy:= round(realy*GDD);
                    PLOTLINE(lstx,lsty,gx,gy);
                End;
        IF DDGRAFX THEN
                Begin
                    dx := round(realx*DDX+lfx);
                    dy := round((1.0-realy)*DDY);
                    DRAW(dx,dy);
                    IF NOT(READIN) THEN DPYUP;
                End;
    End;

PROCEDURE FRAMEDRAW;
  Begin { FRAME }
    realx:=0.0;realy:=0.0;DARKDRAW;    { DRAW FRAME AROUND SCREEN }
    realx:=0.0;realy:=1.0;LIGHTDRAW;
    realx:=1.0;realy:=1.0;LIGHTDRAW;
    realx:=1.0;realy:=0.0;LIGHTDRAW;
    realx:=0.0;realy:=0.0;LIGHTDRAW;
```

```
  End;{ FRAME }

PROCEDURE GODINI;
  Begin { GODINI }
    REWRITE(GODFILE,'TAB.GOD');
    OUT(97);            { DDINIT }
    OUT(1);                     { SCREEN }
    OUTREAL(0.0);
    OUTREAL(0.0);
    OUTREAL(GDD);
    OUTREAL(GDD);
    OUT(4);                     { LITEN }
    BOXGOD:=ASK(11);
    IF BOXGOD THEN FRAMEDRAW;
  End;  { GODINI }

PROCEDURE GODFINI;
 VAR I:integer;
  Begin { DPYUP and KILJOB }
    OUT(17);            { DPYUP }
    OUT(-1);
    OUT(-1);
    FOR I := 1 TO 32 DO OUT(0)  { KILJOB }
  End; { DPYUP and KILJOB }


PROCEDURE DDINI;
  Begin { DDINI }
      PPSET;                    { set up small page printer for text }
      ginit;
      scrset;
      DRKEN;            { dark here so's works like SUN }
      IF not(GODGRAFX) THEN FRAMEDRAW;  { frame if just DD }
  End; { DDINI }

PROCEDURE DDFINI;
  Begin { page printer clear }
      PPDONE;
  End; { page printer clear }



------------------------------------------------------------------------------

{ INITLINELIST, INITVERTLIST, ENTERLINE, ENTERVERTEX, ADDLINEPTR }
Procedure INITLINELIST;
{this initializes the list of lines}
    Begin
        new(11);
        11^.nextline:=nil;
        11^.header:=nil;
        startline:=11;
        tailline:=11;
    End;{INITLINELIST}


Procedure INITVERTLIST;
{this initializes the list of vertices}
    Begin
        new(vl);
        vl^.nextvert:=nil;
        vl^.header:=nil;
        startvert:=vl;
        tailvert:=vl;
    End;{INITVERTLIST}


Procedure ENTERLINE;
{this puts the line into linelist}
```

```
    Begin
        linecount:=linecount+1;
        tailline↑.linenum:=linecount;
        tailline↑.initvert:=verta;
        tailline↑.finalvert:=vertb;
        tailline↑.header:=firstp;
        firstp:=nil;
        new(ll);
        tailline↑.nextline:=ll;
        tailline:=ll;
        tailline↑.nextline:=nil;
    End; {ENTERLINE}


Procedure ENTERVERTEX;
{ starts a new vertex }
    Begin
        tailvert↑.x:=REALX;
        tailvert↑.y:=REALY;
        vertcount:=vertcount+1;
        tailvert↑.vertex:=vertcount;
        tailvert↑.header:=nil;
        new(vl);
        tailvert↑.nextvert:=vl;
        tailvert:=vl;
        tailvert↑.nextvert:=nil;
    End; { ENTERVERTEX }


Procedure ADDLINEPTR(line,vertnum:integer);
{ this adds a line to the linked list of lines associated with a vertex }
    Begin
        vl:=startvert;
        while vl↑.vertex <> vertnum do vl:=vl↑.nextvert;
        lptr:=vl↑.header;
        if lptr = nil then
                begin
                    new(newlptr);
                    vl↑.header:=newlptr;
                end
            else
                begin
                    while lptr↑.nextline <> nil do lptr:=lptr↑.nextline;
                    new(newlptr);
                    lptr↑.nextline:=newlptr;
                end;
        newlptr↑.line:=line;
        newlptr↑.nextline:=nil;
    End; {ADDLINEPTR}



Procedure FINDOLDVERTEX(var vptr:vertlistptr;var oldvert:integer;xold,yold:real);

{ This finds the old vertex which is nearest to the point at which B2 was
   pressed. If the nearest old vertex is further than maxdifference away, it
   tells how far away it is. }

var vertnum:integer;
    tptr:vertlistptr;

    Begin
        tptr:=startvert;
        vptr:=startvert;
        while (tptr<>tailvert) do
        begin
                if DISTANCE(tptr↑.x,tptr↑.y,xold,yold)<DISTANCE(vptr↑.x,vptr↑.y,xold,yold) then
```

```
                    vptr:=tptr;
                tptr:=tptr↑.nextvert;
            end;
            oldvert:=vptr↑.vertex;
            if (DISTANCE(vptr↑.x,vptr↑.y,xold,yold)>maxdifference) then
                writeln(tty,'the nearest old vertex is ',DISTANCE(vptr↑.x,vptr↑.y,xold,yold):4:
2,' away');
        End; {FINDOLDVERTEX}

PROCEDURE AIMVERT(vert:integer);
    Begin
        tptr:=startvert;
        while tptr↑.vertex<>vert do
                tptr:=tptr↑.nextvert;
        REALX:=TPTR↑.X;REALY:=TPTR↑.Y;
    End;

FUNCTION FINDOLDPOINT(var retsptr:linelistptr;TESTX,TESTY:real): BOOLEAN;
  { This finds the old point which is nearest to the point at which B2 was
    pressed. If the nearest old point is further than maxdifference away, it
    tells how far away it is. }

VAR OLDDIST:REAL;


   PROCEDURE SEARCHPOINT(vlptr:linesptr;xold,yold:real);
   Var tstsegnum:integer;
     Begin
        tstsegnum:=ABS(vlptr↑.line);    { get linenumber }
        sptr:=startline;
        while sptr↑.linenum<>tstsegnum do
                sptr:=sptr↑.nextline;
        pptr:=sptr↑.header;
        WHILE pptr<>NIL DO
          Begin
          IF DISTANCE(pptr↑.xp,pptr↑.yp,xold,yold) < OLDDIST THEN
            Begin
                 OLDDIST := DISTANCE(pptr↑.xp,pptr↑.yp,xold,yold);
                 savsptr:=sptr;
            End;
          pptr:=pptr↑.nextpt;
         End;
     End;

    Begin
        AIMVERT(verta);
        savsptr:=nil;
        scanptr:=tptr↑.header;
        OLDDIST:=1.0;
        REPEAT
            SEARCHPOINT(scanptr,TESTX,TESTY);
            scanptr:=scanptr↑.nextline;
          UNTIL scanptr = nil;
        IF OLDDIST > maxdifference THEN
                Begin
{                 WRITELN(TTY, 'No such point on a line from that vertex'); }
                  FINDOLDPOINT:=FALSE;
                End
            ELSE
                Begin
                  verta:=savsptr↑.initvert;
                  vertb:=savsptr↑.finalvert;
                  retsptr:=savsptr;            { copy back for return }
                  FINDOLDPOINT:=TRUE;
                End;
    End; {FINDOLDPOINT}

PROCEDURE [illegible]TEST(saveptr:linelistptr;procedure GRAPHIC; doit:boolean);
```

```
 Begin
       verta:=segptr↑.initvert;
       vertb:=segptr↑.finalvert;
       GRAPHIC;
       AIMVERT(verta); DARKDRAW; { start drawing }
       pptr:=segptr↑.header;READIN:=TRUE;
       while pptr <> nil DO
               Begin
                 REALX:=pptr↑.xp;REALY:=pptr↑.yp;
                 pptr:=pptr↑.nextpt;
                 LIGHTDRAW;
               End;
       AIMVERT(vertb);LIGHTDRAW;READIN:=FALSE;
       DPYUP;DRKEN; { finishing drawing }
 END;

FUNCTION CHECKOLDVERTEX(var retsptr:linelistptr): BOOLEAN;
  Var tstsegnum,checksegnum:integer;
      tstvptr:vertlistptr;
      scnvlptr:linesptr;
      TSTT:BOOLEAN;

    FUNCTION CHECKVERTSOFLINE: BOOLEAN;
      Begin
        tstsegnum:=ABS(scanptr↑.line);  { get linenumber }
        scnvlptr:=tstvptr↑.header;      { scan vertex line list }
        checksegnum:=-1;
        WHILE ((scnvlptr <> NIL) AND (tstsegnum <> checksegnum))
          DO Begin
               checksegnum:=ABS(scnvlptr↑.line);
               scnvlptr:=scnvlptr↑.nextline;
             End;
        IF tstsegnum <> checksegnum THEN
            Begin
              CHECKVERTSOFLINE:=FALSE;
              scanptr:=scanptr↑.nextline;
            End
          ELSE CHECKVERTSOFLINE:=TRUE;
      End;

     Begin
        retsptr:=nil;
        FINDOLDVERTEX(tstvptr,vertb,REALX,REALY);
        savsptr:=nil;   { tptr and tstvptr }
        scanptr:=tptr↑.header;
        TSTT:=FALSE;
        WHILE ((scanptr <> NIL) AND NOT(TSTT))
          DO TSTT:=CHECKVERTSOFLINE;
        IF TSTT THEN
            Begin
                sptr:=startline;
                while sptr↑.linenum<>tstsegnum do
                        sptr:=sptr↑.nextline;
                retsptr:=sptr;
                CHECKOLDVERTEX := TRUE;
            End
          ELSE CHECKOLDVERTEX := FALSE
      End; { CHECKOLDVERTEX }

PROCEDURE DELETESEG(segptr:linelistptr);

  PROCEDURE DELSFROMV(vertnum:integer);
    Begin
      AIMVERT(vertnum);
      scanptr:=tptr↑.header;
      lstscanptr:=tptr↑.header;
      while ABS(scanptr↑.line) <> linenum DO
          Bagin
            lstscanptr:=scanptr;
```

```
          scanptr:=scanptr↑.nextline;
         End;
      IF scanptr=tptr↑.header then tptr↑.header:=scanptr↑.nextline
        ELSE lstscanptr↑.nextline:=scanptr↑.nextline;
    End;

  Begin
    verta:=segptr↑.initvert;
    vertb:=segptr↑.finalvert;
    linenum:=segptr↑.linenum;
    DELSFROMV(verta);            { removeline from start vertex list}
    DELSFROMV(vertb);            { removeline from end vertex list}
    sptr:=startline;
    lstsptr:=nil;
      while sptr <> segptr DO
              Begin
                  lstsptr:=sptr;
                  sptr:=sptr↑.nextline;
              End;
    IF lstsptr=nil THEN startline:=sptr↑.nextline
      ELSE lstsptr↑.nextline:=sptr↑.nextline;
  End; { DELETESEG }

PROCEDURE DELETEPHASE;
var segptr:linelistptr;

PROCEDURE SETDELETION;
  Begin
    FINDOLDVERTEX(tptr,verta,REALX,REALY);
    GETCOORDS;          { select line or vertex point }
    T:=FALSE;
    CASE B1 of
            '1': T:= CHECKOLDVERTEX(segptr);
            '4': T:= FINDOLDPOINT(segptr,REALX,REALY);
         others: Writeln(tty,'1 for vertex; 4 for point')
      End;
    IF T THEN
      Begin
        REMOVETEST(segptr,LITEN,TRUE);
        NEEDCONFIRM:=TRUE;
        writeln(tty,'<B=2> for confirm, <B=8> for reject deletion');
      End
     ELSE writeln(TTY,'Couldn''t delete line (from vertex ',verta:0,')');
  End; { SETDELETION }

PROCEDURE ERRMSG;
  Begin
        writeln(TTY,'B=1 or 4 for deletion, B=2 for confirm, B=8 for reject/quit');
  End;

  Begin { Deletes segments }
    PPSET;
    writeln(TTY,'Select vertex <B=1>, then vertex <B=1> or point on line <B=4>; <B=8> quits.');
    write(TTY,' Once selected, <B=2> confirms deletion, <B=8> cancels it.');
      FINISHED:=FALSE;NEEDCONFIRM:=FALSE;
      REPEAT
        GETCOORDS;          { select vertex }
        CASE B1 OF
         '1': IF NOT(NEEDCONFIRM) THEN SETDELETION
                      ELSE ERRMSG;    { CHOSE THE LINE TO DELETE }
         '2': IF NEEDCONFIRM THEN               { THIS IS CONFIRMATION }
                Begin
                  DELETESEG(segptr);NEEDCONFIRM:=FALSE;
                  writeln(tty,'For deletions B=1 then 1 or 4; B=8 is quit');
                End
              ELSE ERRMSG;
         '8': IF NEEDCONFIRM THEN               { THIS IS REJECTING THE DELETION }
                Begin
                  REMOVETEST(segptr,DRKEN,FALSE);NEEDCONFIRM:=FALSE;
```

```
                 writeln(tty,'For deletions B=1 then 1 or 4; B=8 is quit');
               End
              ELSE Finished:=true;
     OTHERS: ERRMSG
       END;    { OF CASE }

      UNTIL FINISHED;
    T:=TRUE;FINISHED:=FALSE;LAST4:=FALSE;
    writeln(tty,'Returning to entry mode');
  End;{ Deletes segments }

PROCEDURE LASTDELETE;   { last chance at deleting }
  Begin
    PPSET;
    Writeln(TTY,'DELETE? <B=2> for YES, <B=8> for NO');
    T:=FALSE;
    REPEAT
        READTABLET(B1, X4, Y4);
        CASE B1 of
          '2': DELETEPHASE;     { do deletions }
          '8': T:=TRUE;
        others: writeln(TTY,'<B=2> for deletions or <B=8> for none')
        End;
     UNTIL T;
  End;



Procedure STARTOLDVERTEX;

{ This procedure finds an old vertex, enters the number of the incoming
   line in the linked list of lines associated with it, enters the number
   of the old vertex in the line buffer, and calls ENTERLINE to move the
   contents of the buffer into the linked list of lines. It then draws a
   line to the old vertex. }

var linenum,oldvert:integer;

    Begin
        toconfirm:=false;LAST4:=FALSE;
      if jump then
        begin { jump }
            FINDOLDVERTEX(vptr,oldvert,REALX,REALY);
            REALX:=vptr↑.x;REALY:=vptr↑.y;
            verta:=oldvert;
            jump:=false;
            if ddgraph then DARKDRAW;
        end   { jump }
      else
        begin { no jump }
            FINDOLDVERTEX(vptr,oldvert,REALX,REALY);
            vertb:=oldvert;
            REALX:=vptr↑.x;REALY:=vptr↑.y;
            ENTERLINE;
            ADDLINEPTR(linecount,verta);
            ADDLINEPTR(-linecount,vertb);
            if ddgraph then LIGHTDRAW;
            verta:=oldvert;
        end;  { no jump }
    End; {STARTOLDVERTEX}



Procedure STAR[illegible]EX;

{ This proc[illegible] a new entry in the linked list of v[illegible]tices
```

```
   which contains the number and coordinates of the vertex, and adds
   the vertex number to the line buffer. If the vertex is at the end
   of the line (if B4 has been previously pressed to add points to that
   line), it calls ENTERLINE to add the contents of the line buffer
   to the linked list of lines. It also draws a line to the vertex. }

var linenum:integer;

    Begin; {STARTNEWVERTEX}

      toconfirm:=false;LAST4:=FALSE;
      if jump then
        begin { jump }
            ENTERVERTEX;
            verta:=vertcount;
            jump:=false;
            if ddgraph then Begin DARKDRAW;BOX End;
        end   { jump }
      else
        begin { no jump }
            ENTERVERTEX;
            vertb:=vertcount;
            ENTERLINE;
            ADDLINEPTR(linecount,verta);
            ADDLINEPTR(-linecount,vertb);
            if ddgraph then Begin LIGHTDRAW;BOX End;
            verta:=vertcount;
        end;  { no jump }
    End; {STARTNEWVERTEX}




Procedure NEWPOINT;
{ This procedure adds the coordinates of a new point to the linked
   list of points which are associated with a vertex. }
    Begin
        IF toconfirm then DELETEPHASE
          ELSE
            IF NOT(JUMP) THEN
                Begin
                  if firstp=nil then
                          begin
                            new(p);
                            firstp:=p;
                            oldp:=p
                          end
                     else begin
                            new(p);
                            oldp↑.nextpt:=p;
                            oldp:=p;
                          end;
                  p↑.xp:=REALX;p↑.yp:=REALY;
                  p↑.nextpt:=nil;
                  if ddgraph then LIGHTDRAW;
                  LAST4:=TRUE;
                End
               ELSE WRITELN(TTY,'Put the pen down first');
        toconfirm:=false;
    End; {NEWPOINT}



{ UTILITIES }

FUNCTION READINFILE:BOOLEAN;
VAR baseline,basevert,I,numverts,[illegible],numlines,lnum,oldline,COUNT: INTEGER;
```

```
TVAL:BOOLEAN;
tptr:vertlistptr;

Procedure READVERTS;
{ This procedure reads the VERTICES from the vertfile }

Begin
    baseline:=linecount;basevert:=vertcount;
    read(infile,dumchar,dumchar,dumchar);
    read(infile,numverts); { get past 'V =' for N }
    readln(infile);
    FOR I:= 1 to numverts DO
      Begin
        read(infile,vnum,dumchar,dumchar,REALX,dumchar,REALY,dumchar,dumchar);
                    { number, coorda, coordb }
        ENTERVERTEX;
        while not(eoln(infile)) DO
                Begin
                    read(infile,oldline);
                    IF oldline < 0 then signold:=-1 else signold:=1;
                    oldline:=ABS(oldline); oldline:=(oldline+baseline)*signold;
                    ADDLINEPTR(oldline,vnum+basevert);
                End;
        readln(infile);
      End;
    vertcount := basevert + numverts;
    linecount := baseline;
End; { READVERTS }

Procedure READLINES;
{ This procedure reads the LINES from the infile }

    PROCEDURE FINDANDDRAW(vert:integer;procedure FUNC);
        Begin
            tptr:=startvert;
            while tptr↑.vertex<>vert do
                    tptr:=tptr↑.nextvert;
            REALX:=TPTR↑.X;REALY:=TPTR↑.Y;
            FUNC;BOX;
        End;

  Begin
    readln(infile,dumchar,dumchar,dumchar,numlines); { get past 'V =' for N }
    FOR I:= 1 to numlines DO
      Begin
        read(infile,lnum,dumchar,dumchar,verta,dumchar,vertb,dumchar,dumchar);
        verta:=verta+basevert;vertb:=vertb+basevert;
        IF ddgraph THEN FINDANDDRAW(verta,DARKDRAW);
        firstp:=nil;
        while not(eoln(infile)) DO
                Begin
                    read(infile,dumchar,REALX,dumchar,REALY,dumchar);
                    NEWPOINT;
                End;
        IF ddgraph THEN FINDANDDRAW(vertb,LIGHTDRAW);
        ENTERLINE;
        readln(infile);
      End;
  End;{READLINES}

Begin
    TVAL:=ASK(5);
    IF TVAL THEN
        Begin
        PPSET;
{           WRITE(TTY,'Filename:'); }
{           READLN(TTY,filename:QUANT:[ ]); }
{           WRITELN(TTY,QUANT:0); }
            RESET(infile,'NEW.VRT');
```

```
                readin:=true;
                READVERTS;
                READLINES;
            IF (ddgraph and ddgrafx) THEN DPYUP;
            End;
        READINFILE:=TVAL;
    End; { READINFILE }


    Function MAG(X1, Y1, X2, Y2: Real):INTEGER;  { Finds the distance bet. 2 pts. }
        BEGIN
            MAG:= round(Sqrt (Sqr (X2 - X1) + Sqr (Y2 - Y1)));
        END;

    Function ANGLE (X1, Y1, X2, Y2: Real): Real;   { Finds angle theta bet. 2 lines }
    VAR X,Y,xlt:real;
        BEGIN
            X := X2 - X1;
            Y := Y2 - Y1;
            IF abs(X) > abs(y) THEN
                xLT := Arctan (Y/X)
            ELSE xLT := (PI/2) - Arctan (X/Y);
            angle := xlt;
        END;

    FUNCTION INITTABLET:BOOLEAN;
        Begin
            Writeln (tty,'Lower-left corner: ');
                READTABLET(B1, X1, Y1);
            Writeln (tty, 'Top-left corner: ');
                READTABLET(B1, X2, Y2);
            Writeln (tty, 'Top-right corner: ');
                READTABLET(B1, X3, Y3);
            Writeln (tty, 'Lower-right corner: ');
                READTABLET(B1, X4, Y4);
    { these calculations are dumb }
            minx := X1; miny := Y1;
            Theta := ((ANGLE(X1, Y1, X2, Y2) - Pi/2.0) + ANGLE(X2, Y2, X3, Y3) +
                (ANGLE(X4, Y4, X3, Y3) - Pi/2.0) + ANGLE(X1, Y1, X4, Y4))  /4.0;
            pich := MAX(MAG(X1, Y1, X2, Y2),MAG(X4, Y4, X3, Y3));
            picw := MAX(MAG(X1, Y1, X4, Y4),MAG(X2, Y2, X3, Y3));
            Write(tty,'Theta=',theta:5:3,' height=',pich:0,' width=',picw:0);
            Writeln(tty, ' <B=2> confirms, <B=8> rejects');
            READTABLET(B1, X4, Y4);
            CASE B1 of
              '2': INITTABLET:=TRUE;        { make sure happy with this }
              '8': INITTABLET:=FALSE;
            others: writeln(TTY,'2(confirm) or 8(reject)')
             End;
        End;

    PROCEDURE WRITEFILE;
    TYPE bigarray=array[1..500] of integer;
    VAR VTRAN,LTRAN:bigarray;
        VCOUNT,LCOUNT:integer;

      PROCEDURE BUILDVLTABLE;
        Begin
            tptr:=startvert;
            VCOUNT:=0;
            while (tptr<>tailvert) do
              Begin
                verta:=tptr↑.vertex;
                IF tptr↑.header <> nil then
                  Begin
                    VCOUNT:=VCOUNT+1;
                    VTRAN[verta]:=VCOUNT;
                  End
                 ELSE VTRAN[verta]:=-1;     { null }
```

```
          tptr:=tptr↑.nextvert;
          End;
        sptr:=startline;
        LCOUNT:=0;
        while sptr <> tailline do       { no nulls in segs.. they've been removed }
          Begin
            linenum:=sptr↑.linenum;
            LCOUNT:=LCOUNT+1;
            LTRAN[linenum]:=LCOUNT
            sptr:=sptr↑.nextline;
          End;
    End;


Begin   { WRITE OUT VERTICES AND EDGES }
{ write file with entries:
                V = the number of vertices,
                <vertex entries> (N of them),
                L = number of lines,
                <line entries> (M of them).          }
        BUILDVLTABLE;   { make index transforms for vertices and lines }
        writeln(tty,'V = ',VCOUNT:0,', L = ',LCOUNT:0);
{ do Vertices }
        writeln(vertfile,'V = ',VCOUNT:0,' height = ',pich:0,', width = ',picw:0,' (mm)');
        vl:=startvert;
        repeat
          verta:=VTRAN[vl↑.vertex];
          IF verta >0 then write(vertfile,verta:0,':(',vl↑.x:0:3,',',vl↑.y:0:3,'):');
          l:=vl↑.header;
        if l<>nil then
          repeat
            linenum:=l↑.line;if linenum>0 then signold:=1 else signold:=-1;
            linenum:=signold*LTRAN[ABS(linenum)];
            write(vertfile,' ',linenum:0);
            l:=l↑.nextline;
          until l=nil;
          IF verta > 0 then writeln(vertfile);
          vl:=vl↑.nextvert;
        until vl=tailvert;
{ do Lines }
        writeln(vertfile,'L = ',LCOUNT:0);
        ll:=startline;
        repeat
          LINENUM:=LTRAN[ll↑.linenum];
          write(vertfile,LINENUM:0,':<',VTRAN[ll↑.initvert]:0,',',VTRAN[ll↑.finalvert]:0,'>:');
          p:=ll↑.header;
        if p<>nil then
          repeat
            write(vertfile,'(',p↑.xp:0:3,',',p↑.yp:0:3,')');
            p:=p↑.nextpt;
          until p=nil;
          writeln(vertfile);
          ll:=ll↑.nextline;
        until ll=tailline;
End;    { WRITE VERTICES AND EDGES }


------------------------------------------------------------------------------

{MAIN PROGRAM STARTS HERE}

    Begin
        writeln(tty,'Digitizing Tablet; Button Functions:');
        writeln(tty,'   1: start a new vertex (either fresh, or as the end of the current');
        writeln(tty,'      arced segment');
        writeln(tty,'   2: find the nearest vertex to this point, and treat it as the end');
        writeln(tty,'      (or start) of current arced segment');
```

```
    writeln(tty,'   4: take this point');
    writeln(tty,'   8: lift pen;');
    writeln(tty,'      if next is 1 or 2 then CONTINUE drawing');
    writeln(tty,'      if next is 4 then do a DELETION');
    writeln(tty,'      if next is 8 then QUIT');
    DDGRAPH:=ASK(2);
    IF DDGRAPH then
            BEGIN
                DDGRAFX:=ASK(3);
                GODGRAFX:=ASK(4);
                BOXFLAG:=ASK(10);
                IF BOXFLAG THEN
                    Begin
                      write(tty,'  Box size(1,2):');
                      readln(tty);read(TTY,boxsize);
                    End;
                IF GODGRAFX THEN GODINI;
                IF DDGRAFX THEN DDINI;
            END;
    Rewrite(vertfile);
    firstp:=nil;vptr:=nil;vertcount:=0;linecount:=0;
    INITLINELIST;INITVERTLIST;
    REPEAT T:= READINFILE UNTIL NOT(T);
    TABLETIO:=TRUE;DOTAB:=TRUE;
    IF READIN THEN DOTAB:=ASK(12);
    IF DOTAB THEN
      Begin
        write(tty,'TTYline number:');
        readln(tty);read(tty,ttyline);
        bitini(ttyline);
        REPEAT T:=INITTABLET UNTIL T;   { get corners of image }
        PPSET;
        writeln(tty,'*Ready*');
        readin:=false;finished:=false;toconfirm:=false;jump:=true;LAST4:=FALSE;
        firstp:=nil;
        repeat
          GETCOORDS;
            case B1 of
                    '1':STARTNEWVERTEX;
                    '2':STARTOLDVERTEX;
                    '4':NEWPOINT;
                    '8':Begin
                          IF NOT(LAST4) THEN
                            Begin
                              if toconfirm then finished:=true
                                else Begin
                                        toconfirm:=true;
                                        writeln(tty,'*Pen is UP*');
                                     End;
                              jump:=true
                            End
                          ELSE writeln(tty,'put the pen down first');
                        End;
                  others:Writeln(tty,'Garbled data')
            end;
         until finished;
      End;
    IF DDGRAPH THEN
        Begin
          IF GODGRAFX THEN GODFINI ELSE IF DDGRAFX THEN DDFINI;
        End;
    WRITEFILE;
End. {MAIN}
```

```
;code to allow PASCAL to get BITPAD data from TTY other then job's TTY
; Macro-10 assembly code, with local code for TTY driver
        title   trace

--------------------------------------------------------------------------

item==0
sc1==1
sc2==2
sc3==3
sc4==4
p==↑017
MODE==0 ;ASCII
JOBFF==↑0121
chnsts=↑0716000000000
ttyskp=↑0047000400116
PPIOT==↑0702000000000
DBEEP==↑0047000400111           ;BEEP UUO DEFN
DPYCLR==↑0701000000000
INSKIP==↑0051540000000
SLEEP==↑0047000000031
OPDEF PPSEL[PPIOT 0,]
OPDEF PPACT[PPIOT 1,]
OPDEF DPYPOS[PPIOT 2,]
OPDEF DPYSIZ[PPIOT 3,]
OPDEF PPREL[PPIOT 4,]
OPDEF LEYPOS[PPIOT 6,]

        radix   10

        entry   BITINI,BITPAD,PPSET,PPSET2,PPDONE,TTYTAB

BITINI: POP     P,RETAD
        MOVEM   SC2,LINE        ;LINE PARAMETER PASSED BY VALUE
        MOVEI   SC3,↑017
BITLOP: HRRZ    SC1,SC3         ;GET A CHAN TO TEST
        LSH     SC1,18+5
        OR      SC1,[CHNSTS SC2]
        XCT     SC1
        TRNE    SC2,↑0400000    ;USED ALREADY?
          SOJGE SC3,BITLOP      ;TRY UNTIL NONE THERE
        JUMPL   SC3,NOCHNS
        HRRZS   SC3
        LSH     SC3,5
        HRLZM   SC3,CHAN        ;THIS IS THE CHANNEL
        MOVE    ITEM,[TTYSKP]
        IOR     ITEM,CHAN
        MOVEM   ITEM,BITCHK
        MOVE    ITEM,LINE
        SETZ    SC2,
LINLOP: IDIVI   ITEM,10         ;TREAT AS DECIMAL NUMBER (ALTHOUGH IT IS OCTAL)
        ADDI    SC1,↑020
        LSHC    SC1,-6          ;SIDE IT OVER
        JUMPN   ITEM,LINLOP     ;GO TILL DONE
        HLRM    SC2,BITDVC+1    ;LINE IN SIXBIT
BITOPN: MOVE    ITEM,[OPEN BITDVC]
        IOR     ITEM,CHAN       ;MAKE OPEN WORD
        XCT     ITEM
          HALT  BITOPX
        MOVEI   SC1,TTIBFS
        EXCH    SC1,JOBFF
        MOVE    ITEM,[INBUF 2]
        IOR     ITEM,CHAN
        XCT     ITEM
        MOVEM   SC1,JOBFF
```

```
         MOVE    ITEM,[INPUT]
         IOR     ITEM,CHAN
         MOVEM   ITEM,INCMD
         JRST    @RETAD             ;DONE INITING TTY LINE

NOCHNS:  OUTSTR  [ASCIZ/
No channels left.. forget it
/]
         POPJ    P,

TTYSLP:  MOVEI   0,0                ;SLEEP A JIFFY
         SLEEP   0,
TTYTAB:  INSKIP  0                  ;has a char been typed?
           JRST  DOBTPD             ;NO
         SETOM   1(P)               ;return TRUE (SO CALLER GETS INPUT FROM TTY)
         POPJ    P,

DOBTPD:  XCT     BITCHK             ;IF THERES A CHAR IN BITPAD BUFF, DO IT
           JRST  TTYSLP             ;ELSE SLEEP A BIT AND TRY AGAIN
         PUSHJ   P,BITPAD           ;GET INPUT FROM BITPAD
         SETZM   1(P)               ;return FALSE
         POPJ    P,

BITPAD:  POP     P,RETAD
         HRRZM   SC2,RETVL1
         HRRZM   SC3,RETVL2
         HRRZM   SC4,RETVL3
FRC6:    MOVEI   ITEM,6             ;FIRST IS 6 DIGITS
         MOVEM   ITEM,COUNT
         PUSHJ   P,GETBYT
           JRST  FRC6               ;INSIST ON GOOD DATA
         MOVEM   SC1,FSTVAL
         MOVEI   ITEM,5             ;SECOND IS 5 DIGITS
         MOVEM   ITEM,COUNT
         PUSHJ   P,GETBYT
           JRST  FRC6               ;BAD DATA, LOOK FOR 6 BEFOE CONTINUING
INDON:   MOVEM   SC1,@RETVL3        ;RETURN VALUE3 (Y VALUE)
         MOVE    SC1,FSTVAL         ;STRIP OFF BUTTON FROM FSTVAL
         IDIVI   SC1,↑D100000
         MOVEM   SC1,@RETVL1        ;RETURN BUTTON (RETVL1)
         MOVEM   SC2,@RETVL2        ;RETURN VALUE2 (X VALUE)
         JRST    @RETAD

GETBYT:  SETZ    SC1,               ;RETURN VALUE
BITILP:  SOSG    IBF+2              ;COUNTER
           PUSHJ P,REFBUF
         ILDB    SC2,IBF+1
         CAIG    SC2,"9"            ;DONT START TILL GET A DIGIT
         CAIGE   SC2,"0"
           JRST  BITILP
CNTLOP:  IMULI   SC1,10
         ADDI    SC1,-↑O60(SC2)     ;SUM DIGITS
         SOSG    COUNT              ;DIGIT COUNTER
           JRST  GOTCNT
         SOSG    IBF+2              ;BUFF COUNTER
           PUSHJ P,REFBUF
         ILDB    SC2,IBF+1          ;NOW MUST HAVE 5 OR 6 DIGITS (1ST OR 2ND)
         CAIG    SC2,"9"
         CAIGE   SC2,"0"
           JRST  BITBAD             ;GARBLED .. LOOK FOR 6 AGAIN
         JRST    CNTLOP             ;GO FOR REST
GOTCNT:  SOSG    IBF+2              ;BUFF COUNTER
           PUSHJ P,REFBUF
         ILDB    SC2,IBF+1          ;NEXT MUST BE NON-DIGIT
         CAIG    SC2,"9"
         CAIGE   SC2,"0"
           JRST  [illegible]D
BADCLR:  SOSG    I F+2              ;NOW FIND CARRET
           PUSHJ P,REFBUF
```

```
        ILDB    SC2,IBF+1
BITBAD: CAIE    SC2,↑O15        ;CARRET
          JRST  BADCLR          ;NO, CLEAR THE CHARS
        MOVEI   SC1,-1
        DBEEP   SC1,
        OUTSTR  [ASCIZ/Bad data/]
        POPJ    P,              ;YES, RETURN ERROR

GOODEND:AOS     (P)             ;GOOD ONE SKIPS BACK
        POPJ    P,

REFBUF: MOVE    ITEM,INCMD
        XCT     ITEM
        SOS     IBF+2
        POPJ    P,              ;GO BACK WITH NEW DATA

PPSET:  PPSEL   2               ;PAGE PRINTER 2 ON
        PPACT   ↑O100000        ;SEE 2, erase 0
        DPYPOS  -450
        DPYSIZ  ↑O2001          ;2 LINES AT BOTTOM OF SCREEN, SCROLL CONTINUOUSLY
        LEYPOS  0               ;LINE EDITOR AT BOTTOM OF SCREEN
        PPACT   ↑O500000        ;SEE 2 and 0 (now cleared)
        POPJ    P,

PPSET2: PPSEL   2               ;PAGE PRINTER 2 ON
        PPACT   ↑O100000        ;SEE 2, erase 0
        DPYPOS  -400
        DPYSIZ  ↑O4001          ;2 LINES AT BOTTOM OF SCREEN, SCROLL CONTINUOUSLY
        LEYPOS  0               ;LINE EDITOR AT BOTTOM OF SCREEN
        PPACT   ↑O500000        ;SEE 2 and 0 (now cleared)
        POPJ    P,

PPDONE: PPACT   ↑O400000        ;SEE 0 (clear 2)
        DPYCLR                  ;FREE PP 2
        POPJ    P,

BITDVC: EXP     MODE
        SIXBIT/TTY   /
        EXP     IBF
IBF:    EXP     0,0,0
TTIBFS: BLOCK   100
        INTEGER INCMD,CHAN,LINE,RETAD,RETVL1,RETVL2,RETVL3,COUNT,FSTVAL,BITCHK

        END
```

# 10: Epipolar Registration Program

See section 4 for documentation on using this program.

---

```
Begin "epipolar determinations"

    REQUIRE "16A" COMPILER_SWITCHES; comment FIXR,FLTR,ADJSP;
    DEFINE π="3.141592653";
    DEFINE UNLESS="WHILE ¬";
    DEFINE α="COMMENT ";
    DEFINE THRU="STEP 1 UNTIL";
    DEFINE TAB="'11";
    DEFINE CR="'15";
    DEFINE LF="'12";
    DEFINE CRLF="'15&'12";
    DEFINE CCRLF="&'15&'12";
    DEFINE NN=20;

REQUIRE "CAMRAS.HDR[1,jfm]" SOURCE_FILE;
        α Gennery's camera solver;
REQUIRE "MATRIX.HDR[1,JFM]" SOURCE_FILE;
        α Matrix operation utilities;
REQUIRE "CHOSPT.HDR[S,HHB]" SOURCE_FILE;
        α Harlyn's point matcher;
REQUIRE "PIXSAI[S,HE]" LIBRARY;
REQUIRE "PIXFAI[S,HE]" LIBRARY;
        α Source files for image manipulation procedures: see below;
EXTERNAL INTEGER PROCEDURE PIXEL(REFERENCE INTEGER PIX; INTEGER I,J);
        α gets the value PIX of a pixel of coordinates I,J;
EXTERNAL PROCEDURE PUTEL(REFERENCE INTEGER PIX; INTEGER I,J,VALUE);
        α sets the value PIX of a pixel of coordinates I,J;
EXTERNAL INTEGER PROCEDURE PFLDIM(STRING FILNAM);
        α gets the proper array dimension to represent the image called FILNAM;
EXTERNAL INTEGER PROCEDURE GETPFL(STRING FILNAM; REFERENCE INTEGER PICTURE);
        α fills array starting at PICTURE ( for dimension see PFLDIM )
          with picture contained in file FILNAM;
EXTERNAL INTEGER PROCEDURE
         PUTPFL(REFERENCE INTEGER PICTURE; STRING FILNAM; INTEGER MODE(1));
        α writes out a picture file FILNAM with data
          contained in array starting at PICTURE;
EXTERNAL INTEGER PROCEDURE PIXDIM(INTEGER HEIGHT,WIDTH,BITS);
        α gets the proper array dimension to represent a HEIGHT x WIDTH image;
EXTERNAL INTEGER PROCEDURE MAKPIX(INTEGER HEIGHT,WIDTH,BITS;
                                  REFERENCE INTEGER PICTURE);
        α creates an image array, starting at PICTURE, for a  HEIGHT x WIDTH image;
EXTERNAL PROCEDURE WIPE(REFERENCE INTEGER PIX; INTEGER value);
        α makes every pixel of the array equal to value;



α Global variables;

REAL ARRAY GG[1:5];
        α Adjusted values of the parameters (azimuth,elevation,pan,tilt,and roll);
REAL ARRAY  RM[1:3,1:3];
        α rotation matrix between base1 and base 2;
REAL ALPHA,BETA,GAMMA,LAMDA,MU,NU;
        α translation vector components in base 1 and 2;
INTEGER ECASE;
        α ECASE is a flag for the kind of geometry we have
                ECASE=1 : two epipoles ( most common case )
```

```
                ECASE=2 : one epipole in image 1
                ECASE=3 : one epipole in image 2
                ECASE=4 : no epipole;
INTEGER PBITS;
        α number of bits per pixel, returned by CHOSPT;
REAL ARRAY E1,E2[1:2];REAL ARRAY V1,V2[1:2];
        α points and vectors in images 1 & 2,
          epipoles or epipolar directions when they exist,
          else, generic points and vectors;
REAL FF1,FF2;
        α Focal distances in images 1 & 2;
REAL ARRAY M,MINV[1:2,1:2];
        α matrix A of the text and its inverse;
REAL K,N1;
        α  generic and total epiline numbers: 0≤K<N1;
REAL LOX,LOY,LNX,LNY;
        α dimensions of the images( output of CHOSPT) and of the transforms;
INTEGER IK,ILOX,ILOY,ILNX,ILNY;
        α the same, truncated, as integer variables;
REAL THETA,THETA0,THETA1 THETA2;
        α generic,minimum,maximum and increment
        angles under which image 1 is viewed from E1;
REAL PHI0,PHI1;
        α min and max angles for image 2;
REAL D1MIN,D1MAX,D2MAX,D2MIN,D;
        α minimum, maximum and generic distances from epipoles to images 1 & 2;
REAL ARRAY OF3[1:2];
        α   offset for case 3, used in epiline calculation;
INTEGER I;      α generic index;
BOOLEAN RIGHT1,RIGHT2;
        α true if epipole right of image, output of limits;
BOOLEAN DEBUG;
STRING P1,P2;
        α filenames for images P1 and P2;
INTEGER SIZP,SIZS;
        α Size of the arrays that contain original and transformed image;

REAL ARRAY LX[1:NN]; α ARGUMENTS FOR CAMRAS;
REAL ARRAY LY[1:NN];
REAL ARRAY RX[1:NN];
REAL ARRAY RY[1:NN];

INTEGER NUM;



----------------------------------------------------------------------------

α I/O procedures:

INTEGER PROCEDURE GETANSWER;
    Begin INTEGER ANS; ANS←INCHRW LAND '137; PRINT(CRLF); RETURN(ANS) End;

BOOLEAN PROCEDURE ASK(STRING QUESTION);       α ask a yes/no question;
    Begin "ask"
      INTEGER ASK;
        DO BEGIN print(question,"(Y or N)? ");
             ask←getanswer;
             END UNTIL ask="Y" v ask="N";
        return(if ask="Y" then TRUE else FALSE)
    End "ask";

PROCEDURE GETD(REFERENCE INTEGER NUM; STRING QUESTION; INTEGER  EFAULT);
    Begin "GETD"
      PRINT(QUES[illegible] );
      PTOSTR(0,C[illegible]LF));
      NUM ← CVD(      L)
    [illegible]";
```

```
PROCEDURE SPACEPAUSE(STRING TEXT);
    BEGIN
      INTEGER T;
        PRINT(TEXT);
        DO T←INCHRW UNTIL T=" ";
        PRINT(CRLF);
    END;

PROCEDURE GETO(REFERENCE INTEGER NUM; STRING QUESTION; INTEGER DEFAULT);
    Begin "GETO"
      PRINT(QUESTION);
      PTOSTR(0,CVOS(DEFAULT));
      NUM ← CVO(INCHWL)
    End "GETO";

PROCEDURE GETF(REFERENCE REAL RNUM; STRING QUESTION; REAL DEFAULT);
    Begin "GETF"
     STRING NUMSTR;INTEGER BRCHAR;
      PRINT(QUESTION);
      PTOSTR(0,CVF(DEFAULT));
      RNUM ← REALSCAN(NUMSTR←INCHWL,BRCHAR);
    End "GETF";



----------------------------------------------------------------------------

α Arithmetic procedures;

SIMPLE PROCEDURE PROD2 (REAL ARRAY A,B,C);
        α multiplies 2*2 matrices;
    BEGIN
        REAL T;
        INTEGER I,J;
        FOR I←1 STEP 1 UNTIL 2  DO
                BEGIN
                    T←0;
                    FOR J←1 STEP 1 UNTIL 2  DO
                    T ← T + B[I,J]*C[J];
                    A[I]←T
                END
    END;


SIMPLE PROCEDURE PROD3 (REAL ARRAY A,B,C);
        α multiplies 3*3 matrices;
    BEGIN
        REAL T;
        INTEGER I,J;
        FOR I←1 STEP 1 UNTIL 3  DO
                BEGIN
                    T←0;
                    FOR J←1 STEP 1 UNTIL 3  DO
                    T ← T + B[I,J]*C[J];
                    A[I]←T
                END
    END;



PROCEDURE PARAMETERS;
   determines the rotation matrix R and the translation vector t in both bases;
  Begin "PARAMETERS"
    REAL ARRAY TM1[1:3,1:3],T1,T2[1:3];
        α TM1 is translation matrix (matrix A in [illegible])
```

```
      T1 and T2 are translation vectors in base 1 and base 2;
 REAL ARRAY A1,A2,B1,B2,B3,D[1:3,1:3];

 S1←S2←S3←3;
 ROTMAT(A1,D,GG[1],-2);
 ROTMAT(A2,D,GG[2],1);
 ROTMAT(B1,D,GG[3],2);
 ROTMAT(B2,D,GG[4],-1);
 ROTMAT(B3,D,GG[5],-3);
 MULT(TM1,A1,A2);  α translation matrix;
 MULT(D,B2,B1);
 MULT(RM,B3,D);  α rotation matrix;

 ALPHA←T1[1]←TM1[1,3];   α translation vector is 3rd column of translation matrix;
 BETA←T1[2]←TM1[2,3];
 GAMMA←T1[3]←TM1[3,3];

 PROD3(T2,RM,T1);  α T2 is translation vector in second base;

 LAMDA←T2[1];
 MU←T2[2];
 NU←T2[3];

 PRINT("ROTATION MATRIX RM:",CRLF);
 PRINT(RM[1,1],RM[1,2],RM[1,3],CRLF);
 PRINT(RM[2,1],RM[2,2],RM[2,3],CRLF);
 PRINT(RM[3,1],RM[3,2],RM[3,3],CRLF);

 PRINT("TRANSLATION VECTOR t:",CRLF);
 PRINT(ALPHA,BETA,GAMMA,"IN BASE ONE",CRLF);
 PRINT(LAMDA,MU,NU,"IN BASE TWO",CRLF);

end "PARAMETERS";



α Geometric procedures;

SIMPLE PROCEDURE RECT(REAL  EX,EY,VX,VY,RO;REFERENCE REAL X,Y);

α  rect returns the rectangular coordinates of a point with  olar coordinates
   ro and α with respect to a shifted origin e(ex,ey).
   vx and vy are supposed proportional to cosα and sin α, but vx>0:
   α is always in [-π/2,π/2], and RO should have a sign;
  BEGIN "RECT"
      X←SQRT(VX↑2+VY↑2);
      VX←VX/X;
      VY←VY/X;
      X←EX+RO*VX;
      Y←EY+RO*VY;
  END "RECT";


PROCEDURE POLAR( REAL EX,EY,X,Y; REFERENCE REAL THETA,RO);

α  POLAR returns the polar coordinates of a point (x,y) with respect to a
   shifted origin (ex,ey)
   RO is always a positive number,and THETA is in the determination [0,2π];

  Begin "POLAR"
      RO←SQRT((EX-X)↑2+(EY-Y)↑2);
      THETA←ATAN((EY-Y)/(EX-X));
      IF (X-EX)<0 THEN THETA←THETA+π;
      IF THETA <0 THEN THETA←THETA+2*π;
  END "POLAR";
```

```
SIMPLE PROCEDURE NEWX( REAL EX,EY,VX,VY;REFERENCE  REAL NX);
α  A point E has the coordinates EX and EY in a plane where there is a vector V.
   If this plane is rotated so that V becomes the new X-axis, and shifted so
   that the old origin is anywhere on the new Y-axis, the new abscissa of
   E is just its projection onto V:;
    BEGIN; NX←(EX*VX+EY*VY)/SQRT(VX↑2+VY↑2);END;

BOOLEAN PROCEDURE INPICTURE(REAL X,Y,LX,LY);
    RETURN( IF (0≤X≤LX ∧ 0≤Y≤LY) THEN TRUE ELSE FALSE);



PROCEDURE LIMITS( REAL LX,LY,EX,EY;REFERENCE REAL ALPHA0,ALPHA1,DMIN,DMAX;
                  REFERENCE BOOLEAN RIGHT);
α  LIMITS returns:
   1) the minimum and maximum distances DMIN and DMAX from a point
   E to a rectangle of length LX and height LY with bottom left corner at
   origin.( typically, a picture and its epipole )
   2) The minimum and maximum angle through which the rectangle is viewed
   from point E. These angle ALPHA0 and ALPHA1 are in the determination
   [0,2π]( or occasionally, [-π/2,+π/2] )  and the distances are consequently
   positive only.;

  BEGIN "LIMITS"
      REAL X,Y;
      REAL β1,β2,β3,β4;
      DMIN←SQRT(EX↑2+EY↑2);   α initialize;
      DMAX←0;

      FOR X←1 THRU LX DO
              BEGIN
                α  DESCRIBE HORIZONTAL SIDES OF RECTANGLE;
                DMAX←DMAX MAX SQRT((EX-X)↑2+EY↑2) MAX SQRT((EX-X)↑2+(EY-LY)↑2);
                DMIN←DMIN MIN SQRT((EX-X)↑2+EY↑2) MIN SQRT((EX-X)↑2+(EY-LY)↑2);
              END;
      FOR Y←1 THRU LY DO
              BEGIN
                α  DESCRIBE VERTICAL SIDES OF RECTANGLE;
                DMAX←DMAX MAX SQRT(EX↑2+(EY-Y)↑2) MAX SQRT((EX-LX)↑2+(EY-Y)↑2);
                DMIN←DMIN MIN SQRT(EX↑2+(EY-Y)↑2) MIN SQRT((EX-LX)↑2+(EY-Y)↑2);
              END;

      α  angle determination;
      α All angles are initially supposed in [0,2π];
      IF(0≤EX≤LX   ∧ 0≤EY≤LX)THEN BEGIN
              α  epiline is inside the image;
              ALPHA0←0;
              ALPHA1←2*π;
              END
              ELSE BEGIN
              α  e1 is outside the image:
              determine the limit angles by inspecting the corners only;
              POLAR(EX,EY,0,0,β1,D);
              POLAR(EX,EY,LX,0,β2,D);
              POLAR(EX,EY,0,LY,β3,D);
              POLAR(EX,EY,LX,LY,β4,D);
              ALPHA1←(β1 MAX β2 MAX β3 MAX β4);
              ALPHA0←(β1 MIN β2 MIN β3 MIN β4);
              IF ALPHA1-ALPHA0>π THEN BEGIN
              α [0,2π] is not a convenient interval for our angles
                ( the epipole is left of the image, on the same vertical level);
              α Angles are in [-π/2,π/2];
                                        β1←ALPHA1;
                                        ALPHA1←ALPHA0;
                                        ALPHA0←β1-2*π;
      IF DEBUG THEN PRINT("EPIPOLE ON LEFT SIDE AND SAME VERTICAL LEVEL AS IMAGE:",CRLF);
      IF DEBUG THEN PRINT("FOR CONNEXITY, ANGLES ARE NOT CHOSEN IN [0,2π]");
      IF DEBUG THEN PRINT("IN LIMITS, α0= ",ALPHA0,CRLF);
```

```
    IF DEBUG THEN PRINT("IN LIMITS, α1= ",ALPHA1,CRLF);
                                END;
            α  if epipole right of picture, set flag;
            RIGHT←(EX>LX/2);
            α  and invert scanning order by exchanging angles;
             IF RIGHT THEN BEGIN
                           β1←ALPHA1;
                           ALPHA1←ALPHA0;
                           ALPHA0←β1;
                         END;
            END;
  END "LIMITS";



PROCEDURE FRAME( REAL EX,EY,VX,VY,LX,LY;REFERENCE REAL I1X,I1Y,I2X,I2Y;
REFERENCE BOOLEAN OUTFRAME);
α  Given a point E and a vector V ( they define a line ), frame will determine,
   if they exist, the intersections of this line with the frame of the
   rectangle [0:LX,0:LY];

  BEGIN "FRAME"
   REAL X,Y;INTEGER FLAG;

   PROCEDURE ASSIGN;
      BEGIN "ASSIGN"
      IF FLAG=0 THEN BEGIN I1X←X; I1Y←Y; END;
      IF FLAG=1 THEN BEGIN I2X←X; I2Y←Y; END;
      IF FLAG>1 THEN PRINT("ERROR IN PROCEDURE FRAME",CRLF);
      FLAG←FLAG+1;
      END "ASSIGN";

    FLAG←0;

    IF VX≠0 THEN BEGIN
    X←0;
    Y←EY+(X-EX)*VY/VX;
    IF INPICTURE(X,Y,LX,LY) THEN ASSIGN;
    X←LX;
    Y←EY+(X-EX)*VY/VX;
    IF INPICTURE(X,Y,LX,LY) THEN ASSIGN;
    END;

    IF VY≠0 THEN BEGIN
    Y←0;
    X←EX+(Y-EY)*VX/VY;
    IF INPICTURE(Y,X,LY,LX) THEN ASSIGN;
    Y←LY;
    X←EX+(Y-EY)*VX/VY;
    IF INPICTURE(Y,X,LY,LX) THEN ASSIGN;
    END;

    OUTFRAME←(FLAG=0);
    IF DEBUG THEN PRINT (" goes through : ",I1X,I1Y," and through: ",I2X,I2Y,CRLF);
  END "FRAME";



PROCEDURE DETCASE;

α  determines the nature of the epipolar geometry, according to the code:
        case 1: two epipoles E1 and E2
        case 2: one epipole E1
        case 3: one epipole E2
        case 4: no epipoles;
```

```
  Begin "DETCASE"
    REAL THRESHOLD;   α  MINIMUM POSITIVE NUMBER;
      GETF(THRESHOLD,"MINIMUM CONVERGENCE ALLOWED: ",0.0001);
      IF ABS(GAMMA)<THRESHOLD THEN
              IF ABS(NU)<THRESHOLD THEN ECASE←4
                                   ELSE ECASE←3
                        ELSE IF ABS(NU)<THRESHOLD THEN ECASE←2
                                                  ELSE ECASE←1;

      PRINT("                                                  CASE:",ECASE,CRLF);
  end "DETCASE";



PROCEDURE EPIPOLES;
α determines the epipoles and/or the epipolar directions
It also calculates some parameters used by EPILINE ( matrix M and offset OF3 );


  Begin "EPIPOLES"
    REAL DELTA; α Determinant of matrix M;

    IF(ECASE=1 v ECASE=2) THEN BEGIN
                              E1[1]←ALPHA*FF1/GAMMA;
                              E1[2]←BETA*FF1/GAMMA;
                              PRINT("EPIPOLE E1:",E1[1],E1[2],CRLF);
                            END;
    IF(ECASE=1 v ECASE=3) THEN BEGIN
                              E2[1]←LAMDA*FF2/NU;
                              E2[2]←MU*FF2/NU;
                              PRINT("EPIPOLE E2:",E2[1],E2[2],CRLF);
                                  α M is the main conversion matrix;
                                  M[1,1]←NU*RM[1,1]-LAMDA*RM[3,1];
                                  M[1,2]←NU*RM[1,2]-LAMDA*RM[3,2];
                                  M[2,1]←NU*RM[2,1]-MU*RM[3,1];
                                  M[2,2]←NU*RM[2,2]-MU*RM[3,2];

                                α Compute the inverse of matrix M for transformation P2→P1;
                                DELTA←M[1,1]*M[2,2]-M[1,2]*M[2,1];
                                IF DELTA=0 THEN PRINT("UNABLE TO INVERT MATRIX M : CAN'T DUMP
 !") ELSE

                                    BEGIN
                                    MINV[1,1]←  M[2,2]/DELTA;
                                    MINV[2,2]←  M[1,1]/DELTA;
                                    MINV[1,2]← -M[1,2]/DELTA;
                                    MINV[2,1]← -M[2,1]/DELTA;
                                    END;


                                IF ECASE=3 THEN BEGIN
                                  α and OF3 is an offset used in ECASE 3;
                                  OF3[1]←FF1*(NU*RM[1,3]-LAMDA*RM[3,3]);
                                  OF3[2]←FF1*(NU*RM[2,3]-MU*RM[3,3]);
                                  END;
                            END;
    IF(ECASE=3 v ECASE=4) THEN BEGIN
                              V1[1]←ALPHA;
                              V1[2]←BETA;
                              IF V1[1]<0 THEN BEGIN V1[1]←-V1[1];V1[2]←-V1[2] [illegible];
                              PRINT("EPIPOLAR DIRECTION V1:",V1[1],V1[2],CRLF);
                            END;
    IF(ECASE=2 v ECASE=4) THEN BEGIN
                              V2[1]←LAMDA;
                              V2[2]←MU;
                              PRINT("EPIPOLAR DIRECTION V2:",V2[1],V2[2],CRLF);
                            END;
```

```
  End "EPIPOLES";

α more procedures;



PROCEDURE EPILINE;
  Begin "EPILINE";

    IF ECASE=1 THEN BEGIN
        α  This is an epipolar vector in P1;
        THETA←THETA0+(K+0.5)*THETA2;         α  take middle of interval;
        V1[1]←COS(THETA);
        V1[2]←SIN(THETA);
        α make v1 point to the right;
        IF V1[1]<0 THEN BEGIN V1[1]←-V1[1];V1[2]←-V1[2] END;
        α  And this (V2) is the corresponding vector in P2;
        PROD2(V2,M,V1);
        α  make v1 and v2 have consistent directions;
        IF V1[1]*V2[1]<0 THEN BEGIN V2[1]←-V2[1];V2[2]←-V2[2];END;


      END;

    IF ECASE=2 THEN BEGIN
        α  This is an epipolar vector in P1;
        THETA←THETA0+(K+0.5)*THETA2;
        V1[1]←COS(THETA);
        V1[2]←SIN(THETA);
        α  And this is a point belonging to the twin line in P2;
        E2[1]←FF2*(RM[1,1]*V1[1]+RM[1,2]*V1[2])/(RM[3,1]*V1[1]+RM[3,2]*V1[2]);
        E2[2]←FF2*(RM[2,1]*V1[1]+RM[2,2]*V1[2])/(RM[3,1]*V1[1]+RM[3,2]*V1[2]);
      END;

    IF(ECASE=3 v ECASE=4) THEN
          IF ABS(V1[2])*LOX≤V1[1]*LOY THEN BEGIN
                                α  We pick a point on the Y axis;
                                E1[1]←0.0;
                                E1[2]←(LOX*ABS(V1[2]/V1[1])+LOY)*(K+0.5)/N1;
                                IF V1[2]>0 THEN E1[2]←E1[2]-LOX*V1[2]/V1[1];
                              END
                            ELSE BEGIN
                                α  We pick a point on the X axis;
                                E1[1]←(LOY*ABS(V1[1]/V1[2])+LOX)*(K+0.5)/N1;
                                IF V1[2]>0 THEN E1[1]←E1[1]-LOY*V1[1]/V1[2];
                                E1[2]←0.0;
                              END;
    IF ECASE=3 THEN  BEGIN
                       α  V2 directs the twin line in P2;
                       PROD2(V2,M,E1);
                       V2[1]←V2[1]+OF3[1];
                       V2[2]←V2[2]+OF3[2];
                     END;

    IF ECASE=4 THEN BEGIN
                      α  The twin line in P2 goes thru E2;
                      E2[1]←FF2*(RM[1,1]*E1[1]+RM[1,2]*E1[2]+RM[1,3]*FF1)
                               /(RM[3,1]*E1[1]+RM[3,2]*E1[2]+RM[3,3]*FF1);
                      E2[2]←FF2*(RM[2,1]*E1[1]+RM[2,2]*E1[2]+RM[1,3]*FF1)
                               /(RM[3,1]*E1[1]+RM[3,2]*E1[2]+RM[3,3]*FF1);
                    END;
  End "EPILINE";
```

```
  Begin "overhead";
    α DEBUG←ASK("DEBUG"); DEBUG←FALSE;
    IF DEBUG THEN SETPRINT("DEBUG","B");
    IF DEBUG THEN SETFORMAT(0,7) ELSE  SETFORMAT(0,3);
    CHOSPTS(LX,LY,RX,RY,NUM,ILOY,ILOX,PBITS,P1,P2,TRUE);
    IF DEBUG THEN PRINT("NUM=",NUM,CRLF);
    IF DEBUG THEN SETFORMAT(0,7) ELSE  SETFORMAT(0,3);
    LOY←ILOY;LOX←ILOX;
  End "overhead";



PROCEDURE SOLVER;

α  This procedure uses Gennery's code to do the camera registration from
   a  set of corresponding points in the two images. It allows the user
   to pick the points on the images in an interactive way;
Begin "Solver"


    REAL ARRAY X1[1:num],Y1[1:num],X2[1:num],Y2[1:num]; α chosen points;
    REAL ARRAY SXX[1:num],SYY[1:num],SXY[1:num],
            GP[1:5],SP[1:5];
    REAL ARRAY S[1:5,1:5];  α Covariance matrix of the errors in GG;
    REAL ARRAY RESID[1:num,1:3];
    REAL SD,ACC,Q,CONV;
    INTEGER SDP,CORDST;
    INTEGER SDF,MAXEDIT;
    INTEGER NREJECT;
    BOOLEAN ARRAY REJECT[1:num],FLAG[1:7];
    BOOLEAN OUTPUT,RESOUT;

      FOR I←1 THRU NUM DO
          Begin "copy values"
            X1[I]←LX[I];Y1[I]←LY[I];
            X2[I]←RX[I];Y2[I]←RY[I];
          End "copy values";
      FOR I←1 THRU NUM DO
        SXX[I]←SXY[I]←SYY[I]←0;
      GETF(FF1,"Focal length  :",320);    FF2←FF1;
      GETD(MAXEDIT,"Number of points that can be rejected:",0);
        SDP←0.67;CORDST←0; SDF←0 ; OUTPUT←RESOUT←(1=1);
        GP[1]←1.57;GP[2]←GP[3]←GP[4]←GP[5]←0.0;


  CAMERA(FF1,FF2,SDP,CORDST,SDF,NUM,MAXEDIT,OUTPUT,RESOUT,LX,LY,RX,RY,SXX,SYY,SXY,
        GP,SP,GG,S,RESID,SD,ACC,Q,CONV,NREJECT,REJECT,FLAG);    α do the solving;

OUTSTR("Final values of parameters -"&crlf);
PRINT(" azimuth:",CVF(GG[1]),crlf);
PRINT(" elevation:",CVF(GG[2]),crlf);
PRINT(" pan:",CVF(GG[3]),crlf);
PRINT(" tilt:",CVF(GG[4]),crlf);
PRINT(" roll:",CVF(GG[5]),crlf);



End "Solver";




PROCEDURE NEWDIM;

Begin "New_dimensions"
      PRINT(CRLF);
      GETF(N1,"NUMBER OF EPIPOLAR LINES: ",10);
```

```
CASE ECASE OF BEGIN "CASE 1"

    BEGIN END;

    BEGIN
      LIMITS(LOX,LOY,E2[1],E2[2],PHIO,PHI1,D2MIN,D2MAX,RIGHT2);
      IF INPICTURE(E2[1],E2[2],LOX,LOY) THEN LNX←D2MAX+D2MIN
      ELSE LNX←D2MAX-D2MIN;
      LIMITS(LOX,LOY,E1[1],E1[2],THETAO,THETA1,D1MIN,J1MAX,RIGHT1);
      α theta2 is the incremental angle;
      THETA2←(THETA1-THETAO)/N1;
      α the epipole is either inside or outside;
      IF INPICTURE(E1[1],E1[2],LOX,LOY) THEN LNX←D1MAX+D1MIN MAX LNX
      ELSE LNX←D1MAX-D1MIN MAX LNX;
    END;

    BEGIN
      NEWX(LOX,LOY,V2[1],V2[2],LNX);
      LIMITS(LOX,LOY,E1[1],E1[2],THETAO,THETA1,D1MIN,D1MAX,RIGHT1);
      IF INPICTURE(E1[1],E1[2],LOX,LOY) THEN LNX←D1MAX+D1MIN MAX LNX
      ELSE LNX←D1MAX-D1MIN MAX LNX;
    END;

    BEGIN
      NEWX(LOX,LOY,V1[1],V1[2],LNX);
      LIMITS(LOX,LOY,E2[1],E2[2],PHIO,PHI1,D2MIN,D2MAX,RIGHT2);
      IF INPICTURE(E2[1],E2[2],LOX,LOY) THEN LNX←D2MAX+D2MIN MAX LNX
      ELSE LNX←D2MAX-D2MIN MAX LNX;
    END;

    BEGIN
      NEWX(LOX,LOY,V2[1],V2[2],LNX);
      NEWX(LOX,LOY,V1[1],V1[2],LNY);
      LNX←LNX MAX LNY;
    END;
  END "CASE 1";
  LNX←LNX+1;
  LNY←N1;

  α prepare the integers to be used as indexes for arrays;
  IK←K;ILOX←LOX;ILOY←LOY;ILNX←LNX;ILNY←LNY;

  PRINT("THE DIMENSION OF THE NEW PICTURES WILL BE: ",ILNX,"(+4) X ",ILNY,CRLF);

END   "New_dimensions";



PROCEDURE PRINTLINE;

  BEGIN "PRINTLINE"
   BOOLEAN OUTFRAME,MANUAL,PAUSE;
   REAL X1,Y1,X2,Y2;
   INTEGER IX1,IX2,IY1,IY2;

    print(" If you chose the step-by-step option, the execution stops betwean",crlf);
    print(" each line, waiting for you to type Y",crlf);
    print(" If you do not chose this option, the execution proceeds automatically,",crlf);
    print(" either in SLOW or FAST mode",crlf,crlf);
    print("When printing is completed, type Y in both cases to return to the main program."
f);

    MANUAL← ASK(" STEP-BY-STEP PRINTING ");
    IF MANUAL THEN ELSE PAUSE←ASK("SLOW MOTION");
    INITOVERLAY;

    FOR K←0 THRU N1-1 DO
      BEGIN "N1LOOP"
```

```
        IF MANUAL THEN ASK(" LEFT ")
              ELSE IF PAUSE THEN CALL (1,"SLEEP");
        EPILINE;
        FRAME(E1[1],E1[2],V1[1],V1[2],LOX,LOY,X1,Y1,X2,Y2,OUTFRAME);
        IF OUTFRAME THEN PRINT("EPILINE # ",K," OFF PICTURE 1",CRLF)
                    ELSE BEGIN
                          IX1←X1;IX2←X2;IY1←Y1;IY2←Y2;
                          SETLEFT;
                          OVERLINE(IX1,IY1,IX2,IY2);
                         END;

        IF MANUAL THEN ASK ("RIGHT ")
                  ELSE IF PAUSE THEN CALL (1,"SLEEP");
        α PRINT ("     In picture 2, ");
        FRAME(E2[1],E2[2],V2[1],V2[2],LOX,LOY,X1,Y1,X2,Y2,OUTFRAME);
        IF OUTFRAME THEN PRINT("EPILINE # ",K," OFF PICTURE 2",CRLF)
                    ELSE BEGIN
                          SETRIGHT;
                          IX1←X1;IX2←X2;IY1←Y1;IY2←Y2;
                          OVERLINE(IX1,IY1,IX2,IY2);
                         END;
          END "MNLOOP";
        SPACEPAUSE("Type <space> to continue: ");
        FINIOVERLAY;

  END "PRINTLINE";
```

```
PROCEDURE NEWC1( REAL X,Y;REFERENCE REAL Z;REFERENCE INTEGER N);

α  X and Y are the coordinates of a point in picture 2.
  NEWC2 returns N and Z which are the number of the epipolar line( new ordinate)
  and the position of the point on this line ( new abscissa).;

  begin "newc1"
    IF ECASE=1 THEN BEGIN
      POLAR( E1[1],E1[2],X,Y,THETA,Z);
      IF(THETA0≤THETA+π≤THETA1) THEN THETA←THETA+π;
      IF(THETA0≤THETA-π≤THETA1) THEN THETA←THETA-π;
      Z←Z-D1MIN;
      N←N1*(THETA-THETA0)/(THETA1-THETA0)-0.5;
    END ELSE PRINT("SORRY, DEGENERATE CASE NOT YET DEVELOPPED");
  end "newc1";

PROCEDURE NEWC2( REAL X,Y;REFERENCE REAL Z;REFERENCE INTEGER N);

α NEWC2 does the same as newc2 in picture *, but since the epilines are  referenced
  by their position in picture 1, this involve some transformation first.;

  begin "newc2"
    IF ECASE=1 THEN BEGIN
      POLAR(E2[1],E2[2],X,Y,THETA,Z);
      Z←Z-D2MIN;
      V2[1]←COS(THETA);
      V2[2]←SIN(THETA);
      α  And this (V1) is the corresponding vector in P1;
      PROD2(V1,MINV,V2);
      α now we find which angle v2 correspond to and what epiline that is;
      THETA←ATAN(V1[2]/V1[1]);
      IF(THETA0≤THETA+π≤THETA1) THEN THETA←THETA+π;
```

```
      IF(THETA0≤THETA-π≤THETA1) THEN THETA←THETA-π;
      N←N1*(THETA-THETA0)/(THETA1-THETA0)-0.5;
    END ELSE PRINT("SORRY, DEGENERATE CASE NOT YET DEVELOPPED");
  end "newc2";
```

---

```
PROCEDURE TRANSFORM;
  BEGIN "TRANSFORM"

   BOOLEAN OUTFRAME;
   REAL X1,Y1,X2,Y2;
   INTEGER IX1,IX2,IY1,IY2,NX,NN1,VALUE;
   SAFE INTEGER ARRAY PP[0:SIZP-1],SS[0:SIZS-1];  α INPUT AND OUTPUT ARRAYS;
   INTEGER XMIN,XMAX,N;     α Length info;
   STRING EP1,EP2; α transforms filenames;

  PROCEDURE CLEARARRAY;
   BEGIN
    FOR IK←0 THRU N1-1 DO
    FOR I← 0 THRU ILNX-1 DO
        PUTEL(SS[0],IK,I,0);
   END;

  PROCEDURE TRLINE( INTEGER IM; REAL EX,EY,DMAX,DMIN;BOOLEAN INVERT );

    BEGIN "TRLINE"
      REAL LOWERD,UPPERD; α PICTURE BOUNDS;
      REAL VX,VY;
        IF DEBUG THEN PRINT("EPILINE # ",K,CRLF);
        EPILINE;
        IF IM=1 THEN BEGIN VX←V1[1];VY←V1[2];END
                ELSE BEGIN VX←V2[1];VY←V2[2];END;
        FRAME(EX,EY,VX,VY,LOX,LOY,X1,Y1,X2,Y2,OUTFRAME);
        IF OUTFRAME THEN PRINT("EPILINE # ",K," OFF PICTURE ",IM,CRLF)
                    ELSE BEGIN
                          IK←K;IX1←X1;IX2←X2;IY1←Y1;IY2←Y2;
                          IF DEBUG THEN ELSE OVERLINE(IX1,IY1,IX2,IY2);
                          XMIN←DMAX;XMAX←0;       α Initialize;
                          IF INVERT THEN BEGIN UPPERD←-DMIN;LOWERD←-DMAX;END
                                    ELSE BEGIN UPPERD←DMAX; LOWERD←DMIN; END;
                          FOR D← LOWERD THRU UPPERD DO
                              BEGIN "pt-by-pt"
                                RECT(EX,EY,VX,VY,D,X1,Y1);
                                IF INPICTURE(X1,Y1,LOX,LOY) THEN
                                        BEGIN "transfer"
                                          IX1←X1;IY1←Y1; α  coordinates on source image;
                                          VALUE←PIXEL(PP[0],IY1,IX1);
                                          IF DEBUG THEN OVERLINE(IX1,IY1,IX1,IY1);
                                          NX←D-LOWERD;
                                          XMIN←XMIN MIN NX;
                                          XMAX←XMAX MAX NX;
                                          PUTEL(SS[0],IK,NX+4,VALUE);
                                        END "transfer";
                              END "pt-by-pt";


                          N←XMIN LSH -8;    α    8 MSBs of boginning of line;
                          PUTEL(SS[0],IK,0,N);
                          N←XMIN LAND '377;          α 8 LSBs of beginning of line;
                          PUTEL(SS[0],IK,1,N);
                          N←XMAX LSH -8;    α    8 MSBs of end of line;
                          PUTEL(SS[0],IK,2,N);
                          N←XMAX LAND '377;     ·    α 8 LSBs of end of line;
                          PUTEL(SS[0],IK,3,N);
```

```
---------------------------------------------------------------------------

α Start of main program;
BEGIN"Main"
OVERHEAD;
DO  BEGIN
    SOLVER;
    PARAMETERS;
    DETCASE;
    EPIPOLES;
    NEWDIM;

    IF ASK("TRANSFORM THE IMAGES INTO EPIPOLAR SPACE ") THEN
        IF ECASE=1 THEN BEGIN
                            SIZP←PFLDIM(P1);
                            SIZP←PFLDIM(P2) MAX SIZP;
                            ILNX←ILNX+4;    α make space for line length info;
                            SIZS←PIXDIM(ILNY,ILNX,PBITS);
                            TRANSFORM;
                          END
        ELSE PRINT("NO TRANSFORMATION FOR DEGENERATE CASE")
    ELSE IF ASK("DRAW EPIPOLAR LINES ") THEN PRINTLINE;

    IF ASK("DUMP THE PARAMETERS ") THEN DUMP;

    END UNTIL ASK("ANOTHER TRY ")=FALSE;

END "Main";


End "epipolar determinations";
```

```
                    END;

END "TRLINE";



a body of procedure transform;

a DEBUG←ASK("DEBUG");


PRINT("NEW LEFT PICTURE: ");
EP1←INCHWL;
PRINT("NEW RIGHT PICTURE: ");
EP2←INCHWL;

PRINT("PICTURE 1:",CRLF);
INITOVERLAY;
SETLEFT;
GETPFL(P1,PP[0]);
MAKPIX(ILNY,ILNX,PBITS,SS[0]);
FOR K←0 THRU N1-1 DO TRLINE(1,E1[1],E1[2],D1MAX,D1MIN,RIGHT1);
PUTPFL(SS[0],EP1);
FINIOVERLAY;

INITOVERLAY;
PRINT("PICTURE 2:",CRLF);
SETRIGHT;
GETPFL(P2,PP[0]);
MAKPIX(ILNY,ILNX,PBITS,SS[0]);
CLEARARRAY;
FOR K←0 THRU N1-1 DO TRLINE(2,E2[1],E2[2],D2MAX,D2MIN,RIGHT2);
PUTPFL(SS[0],EP2);
SPACEPAUSE("Type <space> to continue: ");
FINIOVERLAY;

END "TRANSFORM";



PROCEDURE DUMP;

  BEGIN "DUMP"


    SETFORMAT(0,7);
    SETPRINT(NULL,"B");

    PRINT("ECASE= ",ECASE,CRLF);
    PRINT("N1= ",N1,CRLF);
    PRINT("E1[1]= ",E1[1],CRLF);
    PRINT("E1[2]= ",E1[2],CRLF);
    PRINT("E2[1]= ",E2[1],CRLF);
    PRINT("E2[2]= ",E2[2],CRLF);
    PRINT("THETA0= ",THETA0,CRLF);
    PRINT("THETA1= ",THETA1,CRLF);
    PRINT("D1MIN= ",D1MIN,CRLF);
    PRINT("D2MIN= ",D2MIN,CRLF);
    PRINT("MINV[1,1]= ",MINV[1,1],CRLF);
    PRINT("MINV[1,2]= ",MINV[1,2],CRLF);
    PRINT("MINV[2,1]= ",MINV[2,1],CHLF);
    PRINT("MINV[2,2]= ",MINV[2,2],CRLF);

    SETPRINT(NULL,"T");

  END "DUMP";
```

# 11: References

[Arnold 1980] Arnold, R.D., and T.O. Binford, "Geometric Constraints in Stereo Vision," *Soc. Photo-Optical Instr. Engineers*, **Vol.** 238, Image Processing for Missile Guidance, 1980, 281–292. (*cited on pp.* 8,9,27,30)

[Arnold 1983] Arnold, R. David, "Automated Stereo Perception," Department of Computer Science, Stanford University, Ph.D. thesis, 1983. (*pp.* 27)

[Baker 1981] Baker, H. Harlyn, "Depth from Edge and Intensity Based Stereo," University of Illinois, Ph.D. thesis, September 1981. (*pp.* 1,27,28,30,31)

[Binford 1981] Binford, Thomas O., "Inferring Surfaces from Images," *Artificial Intelligence*, **Vol.** 17(1981), August 1981, 205–244. (*pp.* 8)

[Brooks 1981] Brooks, Rodney A., ' Symbolic Reasoning Among 3-D Models and 2-D Images," Stanford Artificial Intelligence Laboratory, AIM-343, June 1981. (*pp.* 5,41)

[Duda 1973] Duda, R.O. and P.E. Hart, **Pattern Classification and Scene Analysis**, Wiley, New York, 1973. (*pp.* 9)

[Forney 1973] Forney, G. David Jr., "The Viterbi Algorithm," *Proceedings of the IEEE*, **Vol.** 61, **No.** 3, March 1973, 268–278. (*pp.* 30)

[Gennery 1980] Gennery, Donald B., "Modelling the Environment of an Exploring Vehicle by Means of Stereo Vision," Ph.D. thesis, Stanford Artificial Intelligence Laboratory, AIM 339, June 1980. (*pp.* 2,12,14,27)

[Gimel'farb 1972] Gimel'farb, G.L., V.B. Marchenko, and V.I. Rybak, "An Algorithm for Automatic Identification of Identical Sections on Stereopair Photographs," *Kybernetica* (translations) **No.** 2, March-April 1972, 311–322. (*pp.* 30)

[Hallert 1960] Hallert, Bertil, *"Photogrammetry, Basic Principles and General Survey,"* McGraw-Hill Book Company Inc., 1960. (*pp.* 12,29)

[Hannah 1974] Hannah, Marsha Jo, "Computer Matching of Areas in Stereo Images," Ph.D. thesis, Stanford Artificial Intelligence Laboratory, AIM–239, July 1974. (*pp.* 27)

[Henderson 1979] Henderson, Robert L., Walter J. Miller, C.B. Grosch, "Automatic Stereo Recognition of Man-Made Targets," *Society of Photo-Optical Instrumentation Engineers*, **Vol.** 186, *Digital Processing of Aerial Images*, August 1979. A more complete description is available as: "Geometric Reference Preparation Interim Report Two: The Broken Segment Matcher," Henderson, R.L., Rome Air Development Centre, Rome, New York, RADC–TR–79–80, April 1979. (*pp.* 30)

[Liebes 1981] Liebes Jr., S., "Geometric Constraints for Interpreting Images of Common Structural Elements: Orthogonal Trihedral Vertices," *Proceedings of the DARPA Image Understanding Workshop*, 1981. (*pp.* 6)

[Lowe 1982] Lowe, David G., "Segmentation and Aggregation," *Proceedings of the DARPA Image Understanding Workshop*, Stanford University, September 1982. (*pp.* 3,8)

[Marimont 1982] Marimont, David H., "Segmentation in ACRONYM," *Proceedings of the DARPA Image Understanding Workshop*, Stanford University, September 1982. (*pp.* 1,2,3,31,32,41)

[Moravec 1980] Moravec, Hans P., "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," Stanford Artificial Intelligence Laboratory, AIM-340, Ph.D. thesis, September 1980. (*pp.* 12,27)

[Perkins 1968] Perkins, "Cubic Corners," in Quarterly Progress Report, MIT Electronics Laboratory, April 15,1968 (*pp.* 7)

[Quam 1971] Quam, Lynn H., "Computer Comparison of Pictures," *Stanford Artificial Intelligence Laboratory, AIM-144*, Ph.D. thesis, 1971. (*pp.* 27)

DATE

ILME